uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática
Curso académico 2016-2017

Trabajo de Fin de Grado

"Estudio y evaluación de los sistemas RGB-D SLAM"

Ángel Palacio López

Tutor:

Luis Moreno Lorente

En Leganés, a 22 de septiembre de 2017

Agradecimientos

Quiero hacer constar mi agradecimiento a todo el personal de la Universidad que ha contribuido de una u otra manera a que mi etapa universitaria sea lo más fructífera posible, incluyendo profesores, personal de laboratorio, biblioteca, directiva, administración y mantenimiento entre otros.

A mi familia por el apoyo moral y económico, por su comprensión en los momentos de fracaso y por las esperanzas puestas en mí.

Al tutor, Luis Moreno, y al resto del departamento de Sistemas y Automática por su orientación y ayuda durante el desarrollo del proyecto.

A la comunidad investigadora por publicar de manera accesible y gratuita sus progresos, permitiendo el avance de los sistemas y promoviendo la cooperatividad entre equipos de desarrolladores que se encuentran a miles de kilómetros unos de otros. En particular a Alejo Concha y Javier Civera, los creadores de RGBDTAM, trabajo de referencia en este proyecto.

Por el mismo motivo, al equipo Computer Vision Group de la Universidad de Freiburg, que vela más que nadie por el progreso de tecnologías relacionadas con la visión por computador.

Resumen

El presente proyecto tiene como objetivo plasmar un estudio en profundidad del fundamento y estado del arte de la tecnología SLAM (Localización y Mapeo Simultáneo). Tomando como caso de estudio la variante más utilizada hoy en día en investigación, el SLAM basado en datos de color y profundidad, se dará en primer lugar una visión global de la utilidad de esta técnica y de sus aplicaciones actuales y posibles. Más adelante se desglosarán los componentes del algoritmo para estudiar qué fortalezas y debilidades tiene, poniendo especial énfasis en la comparativa entre el filtro de Kalman como método tradicional y contrastado, y la odometría visual como nueva tecnología aplicable y todavía en desarrollo.

Para ilustrar el estudio, se ha elegido la herramienta RGBDTAM como ejemplo del estado del arte de los sistemas SLAM más eficientes. Se trata de un software RGB-D SLAM puntero desarrollado por varios estudiantes e investigadores de la Universidad de Zaragoza. Se explicará cada uno de los algoritmos y tecnologías que han sido elegidos y su motivo, relacionándolo con los apartados anteriores para entender su contexto.

Finalmente, para darle sentido al estudio se realizarán una serie de pruebas con esta herramienta. Con estas pruebas se busca demostrar la utilidad del SLAM además de buscar los límites a los que se enfrentan los desarrolladores hoy en día. De esta manera se podrá elaborar una pequeña propuesta de hacia dónde deberá evolucionar el algoritmo para mejorar en su rendimiento y eficiencia.

Abstract

The objective of the present project is to show a deep study about the state of the art and the principles behind SLAM technology (Simultaneous Localization And Mapping). The most commonly used type of SLAM in investigation will be the main case of study, which is SLAM based in color and depth data. First of all, a global view of the methods involved in RGB-D SLAM systems will be given, followed by its current and potential applications. After that the subsystems of the algorithm are explained in more detail, so its strenghts and weaknesses can be analysed. Special emphasis will be put on the explanation of the Kalman Filter, as the core of the algorithm in traditional methods, so it can be compared with the visual odometry which is the newest alternative and still to reach its full potential.

In order to ilustrate the study, the model chosen to implement SLAM has been RGBDTAM, a tool created by two investigators from Universidad de Zaragoza. Their work is a reference in efficiency among all the latest RGB-D SLAM programs. Each and every one of the algorithms and technologies chosen to design this software will be explained along with the reason to select it, comparing it with the theory explained previously so the context can be understood.

Finally, in order to give sense to the study, a series of tests will be run to analyse the performance of RGBDTAM. This way we can learn about the purpose of a tool like this, and also about the limits investigators are facing nowadays. After all this, a small proposal on how this technology should be expected to evolve can be made.

Índice

Índice de figuras

Listado de acrónimos

BA: Bundle Adjustment

CNN: Convolutional neural network

CPU: Central Processing Unit

DE: Differential Evolution

EKF: Extended Kalman Filter

g2o: General Graph Optimization

GPS: Global Positioning System

GPU: Graphics Processing Unit

ICP: Iterative Closest Point

KF: Kalman Filter

ORB: Oriented FAST, Rotated Brief

OS: Operating system

PTAM: Parallel Tracking And Mapping

RA: Realidad Aumentada

RAM: Random Access Memory

RANSAC: RANdom SAmple Consensus

RGB: Red, Green, Blue.

RGB-D: RGB + Depth

RGBDTAM: RGB-D Tracking And Mapping

ROS: Robot Operating System

SfM: Structure from Motion

SLAM: Simultaneous Localization And Mapping

SURF: Speeded-Up Robust Features

SW: Software

VSM: Visual SLAM Measurement

CAPÍTULO 1. Introducción a SLAM y su historia

En este primer capítulo se quiere dar una visión general de los fundamentos del SLAM y de qué problemas puede resolver. En primer lugar se explica en qué consiste el problema a resolver y en qué principios se ha de basar cualquier sistema para solucionarlo, a continuación de lo cual se enunciarán las fases del proyecto.

Una vez hecho esto el resto del documento se estructura como sigue: en el capítulo 2 se enuncian los tipos de formas de resolver el problema, asociándolos a sus aplicaciones y analizando el contexto de éstas dentro del estado del arte de la tecnología. El capítulo 3 se dedica al completo a las formas más habituales de implementar SLAM con cámaras de color y profundidad, lo que servirá de pie al capítulo 4 para explicar a fondo un caso concreto de este tipo de SLAM que se ha elegido para hacer experimentos y entender el desarrollo y componentes de una herramienta de esta clase. Estos experimentos se detallan en el capítulo 5 y en el 6 se relatarán las conclusiones extraídas de lo aprendido con los experimentos y con el trabajo. El trabajo se concluye con el séptimo capítulo, un ensayo sobre las implicaciones sociales y económicas de la presencia de SLAM en cada vez más dispositivos robóticos.

1.1. Qué es SLAM

SLAM (Simultaneous Localization And Mapping – localización y mapeo simultáneo) es un proceso por el cual un robot puede construir un mapa de un determinado entorno y al mismo tiempo navegar por dicho entorno. Para ello, se han de identificar o conocer previamente unos puntos característicos que serán usados de referencia por el sistema.

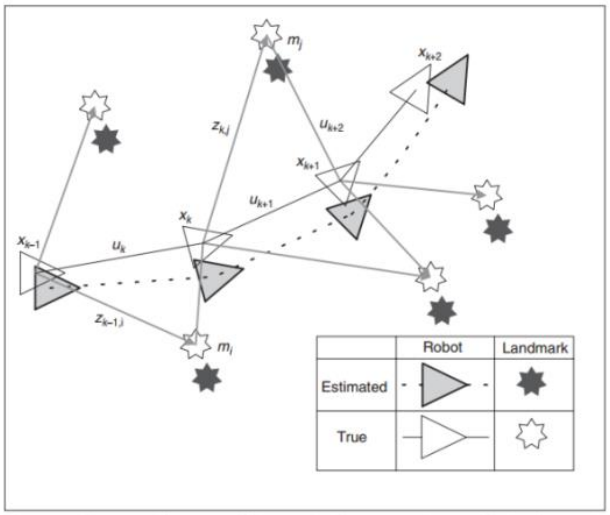


Figura 1: Funcionamiento básico de SLAM

En la figura 1 se puede ver rápidamente el comportamiento real de un robot con una determinada imprecisión en sus sensores, lo que genera incertidumbre en la posición observada de los puntos característicos y por tanto en la idea que tiene el robot de su posición. Aunque los sensores fueran ideales, esta incertidumbre siempre existe al no conocerse la posición de partida del robot. No conocer esta posición de partida es precisamente lo que distingue a SLAM de otras estrategias de localización, ya que no requerir un protocolo de inicialización es una gran ventaja a la vez que una dificultad añadida.

Otra de las cualidades más importantes de SLAM es que las estimaciones de las posiciones de los puntos característicos mejoran conforme se reobservan estos puntos, por lo que con el tiempo el algoritmo va ganando en eficiencia y seguridad.

Se hará referencia a SLAM como “el sistema” y no como “el algoritmo”, ya que son varias etapas las que componen el proceso, cada una con sus propios algoritmos. La solución al problema del SLAM no es por tanto única, siendo muy diversos los enfoques que se pueden tomar para encontrar una solución, como se verá más adelante.

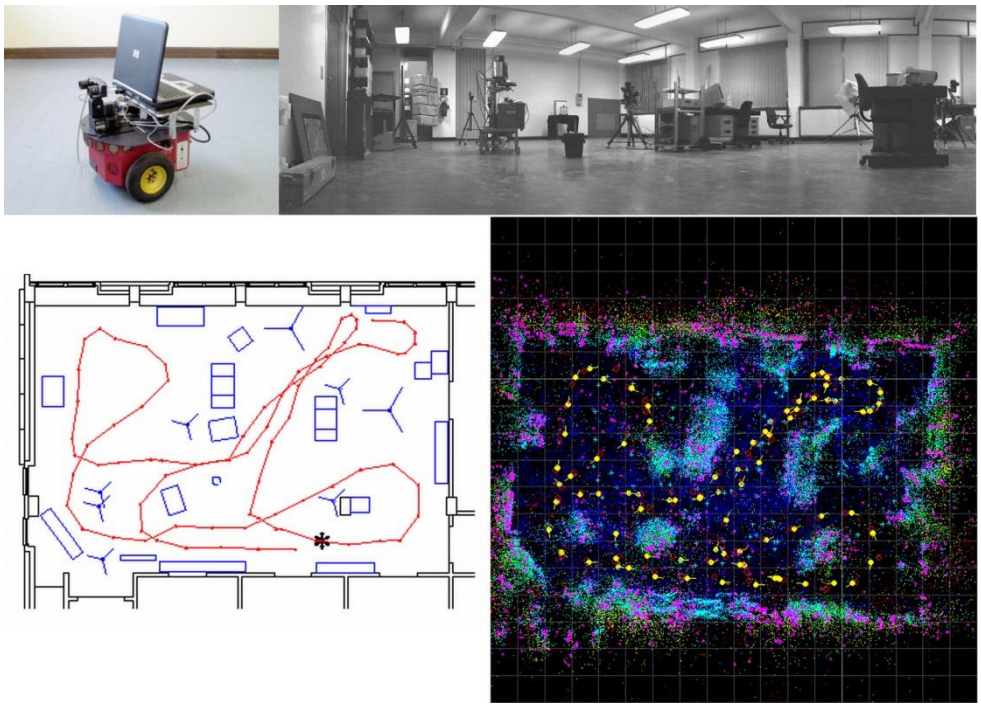


Figura 2: Ejemplo de un experimento y los resultados esperados

Para poder crear los mapas, el algoritmo necesita información de entrada de los sensores adecuados. Como mínimo, es necesario tener datos de profundidad y del comando de movimiento que se efectúa sobre el robot. La odometría fue durante muchos años el método más habitual por ser el más económico, basado en la lectura de las revoluciones de las ruedas del robot y en sensores de inercia para estimar los giros. Hoy en día sin embargo, SLAM es un problema resuelto con frecuencia mediante visión por computador.

1.2. Origen y planteamiento del problema

La necesidad de resolver este problema surge del deseo de mayor autonomía en robots móviles. Una vez éstos puedan moverse de manera independiente por un determinado entorno, podrá dependerse de ellos para tareas cada vez más complejas.

Las primeras publicaciones de investigación sobre SLAM aparecen en California en 1986 [29] (aunque este acrónimo no se acuña hasta 1991 en [3]) y utilizan robots con sensores de laser o sónar junto a información de odometría y dirección por inercia. Debido a ello, los entornos en los que podían navegar debían ser planos y con obstáculos simples y en todo caso conocidos previamente.

En esta época los mundos de la robótica y la inteligencia artificial estaban empezando a unirse, gracias a la aplicación por vez primera de los métodos probabilísticos de estimación a problemas derivados de la robótica. El mayor riesgo de asumir una alta incertidumbre no es sólo obtener una trayectoria falsa, sino confundir puntos característicos, lo cual puede llevar a imprecisiones fatales. En la figura 2 se muestra gráficamente dicho caso.

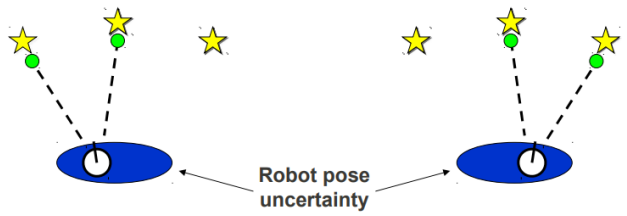


Figura 3: Incertidumbre en la pose del robot

Durante los años siguientes se publicaron estudios clave en este campo que consiguieron el objetivo de manipular la incertidumbre geométrica de manera estadística, lo que significa que con sucesivos escaneos se reduce dicha incertidumbre. La solución más frecuentemente adoptada se basa en modelar un espacio de estados con ruido gaussiano, siendo la implementación más adecuada el Filtro Extendido de Kalman (EKF). En el capítulo 3 se detallará esta solución.

A lo largo de la última década se ha visto un enorme crecimiento en el rendimiento de las soluciones de SLAM que se han ido elaborando. Aunque la mayoría de las propuestas se centraban en mejorar la eficiencia computacional intentando mantener la precisión en la elaboración de mapas, muchos otros encontraron la solución en mejorar el origen de los datos. Cobra aquí especial importancia la incorporación al sistema de la odometría visual, es decir, la obtención del dato de variación de posición del punto de vista a partir de imágenes sucesivas. Aunque más adelante se verá que las cámaras tienen importantes inconvenientes en determinados entornos, a nivel general una buena implementación de odometría visual es más fiable que la odometría clásica. Este tipo de sensor fue implementado de manera exitosa por primera vez en 2003, y es el más utilizado en investigación a día de hoy gracias a la reducción de precio que han ido sufriendo las cámaras RGB y a la aparición de sensores RGB-D con lecturas de color y distancia integradas a coste razonable y con soporte para desarrolladores, como Microsoft Kinect o Asus Xtion.

Desde un punto de vista conceptual y teórico, como ya se ha visto, SLAM es un problema que puede considerarse resuelto. Sin embargo, siguen existiendo áreas con amplio margen de mejora como la construcción del mapa y su uso, o la falta de polivalencia y adaptabilidad en la elección de puntos característicos. Esto último es particularmente importante, ya que existen varios criterios para ubicarlos, lógicamente un robot entrenado para detectar puntos de referencia basados en color no se orientará adecuadamente en una habitación monocromática. Otra cuestión sin una solución estable aún hoy es reconocer cuando el entorno cambia.

El esquema de la figura 4 resume de manera genérica los elementos habituales de un sistema SLAM.

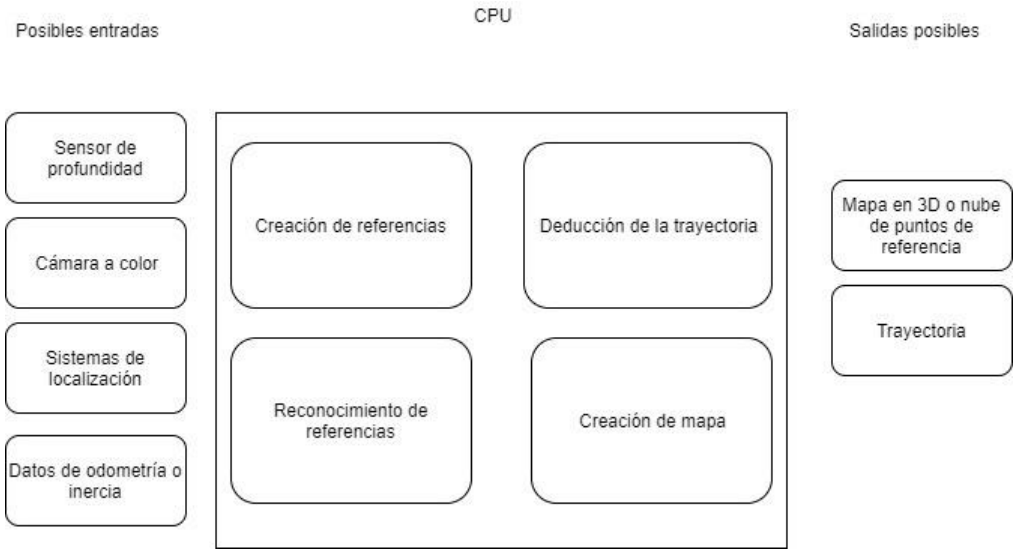


Figura 4: SLAM a alto nivel

Un problema unido a la mejora de rendimiento es mantener la operación en tiempo real. Cuanto mayor es la complejidad del sistema más difícil será que el equipo pueda procesar todos los fotogramas o *frames* que se capten, por lo que es siempre una prioridad mantener lo más bajo posible el coste computacional.

1.3. Motivación y objetivo

El interés principal de este trabajo radica en el estudio de una tecnología que en la última década ha experimentado una revolución. La mejora de las técnicas de fabricación y diseño de hardware permiten ahora un rango de aplicaciones de SLAM impensable antes, lo que ha significado también un aumento en la complejidad de los algoritmos y en las arquitecturas del sistema. Aquí encontramos otro punto de interés fundamental. SLAM integra aplicaciones de familias de algoritmos muy variadas en su estructura, lo que ha hecho necesaria una investigación exhaustiva de todas estas tecnologías para poder entender a fondo el caso de estudio que se elegirá.

El objetivo es, por lo tanto, demostrar que se ha elegido un tema donde los conocimientos adquiridos durante el grado servirán como base a la investigación para obtener una experiencia lo más fructífera posible, gracias a su complejidad, a su actualidad y a sus nuevas aplicaciones.

1.4. Fases del proyecto

Tarea	Descripción	Plazo
Familiarización con el concepto	Durante las primeras reuniones se establece un objetivo genérico de estudiar la tecnología SLAM 2D	½ mes
Concretar la rama de la tecnología a estudiar	Se decide que es más actual e interesante estudiar SLAM en 3D	1 mes
Investigación sobre publicaciones recientes relacionadas	Fase de búsqueda de publicaciones clave y estado del arte	1 mes
Establecer un caso de estudio	Por disponibilidad de Kinect y facilidad para contactar con compañeros con experiencia, se elige el caso de estudio RGB-D SLAM	1 semana
Elección de un proyecto existente del caso de estudio como punto de partida.	Por recomendación de compañeros estudiantes de doctorado especializados, se elige el sistema a analizar, RGBDTAM.	1 semana
Estudio del funcionamiento del programa elegido	Lectura y análisis en profundidad de documentación asociada al programa y de sus predecesores	2 meses
Experimentación y análisis de dicho programa	Análisis de la operación y búsqueda de puntos débiles	2 meses
Finalización	Extracción de conclusiones y redacción del proyecto	1 mes

Tabla 1: planificación del proyecto

CAPÍTULO 2. Actualidad y estado del arte

Esta sección estará dedicada al estado actual del SLAM. Se hablará de qué métodos de implementación son los más populares, qué aplicaciones tienen y qué evolución cabe esperar en el futuro. Este apartado resulta clave para entender el contexto de la realización de este trabajo y la importancia de la tecnología que incorpora SLAM.

2.1. Tipos de SLAM

Existen una gran variedad de características por las que clasificar los sistemas SLAM. Los criterios de los siguientes subapartados se basan en los tipos de subsistemas que integran un sistema completo SLAM, en la disponibilidad de sensores o en las funcionalidades que aporta.

2D o 3D

Los sistemas SLAM de mapeo 2D son útiles sólo en entornos planos y basan sus observaciones en sensores de distancia o profundidad. El algoritmo más popular para resolver este problema se conoce como Grid Mapping (mapeo en rejilla o cuadrícula) y para ello divide el área de trabajo en una cuadrícula de la resolución deseada, donde cada celda tiene una probabilidad de 0 a 1 de estar ocupada. Combinando observación y métodos probabilísticos se va descifrando cada una de dichas celdas.

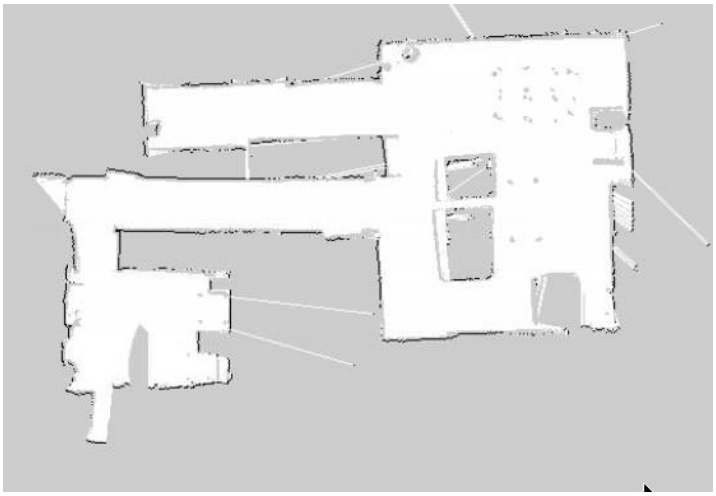


Figura 5: Captura de SLAM 2D del tutorial de SLAM de Christopher J. Tralie

Basado en puntos característicos o directo

Los puntos característicos son referencias que el sistema va a utilizar para ubicarse. Pueden ser datos manuales, balizas que se detectan con un determinado sensor, o creadas por el sistema. Este último caso es el más delicado y menos robusto, debido a que estos puntos

creados llevan asociada ya una incertidumbre. Como se verá más adelante, existen varios criterios visuales para elegir los puntos característicos como la detección de esquinas, formas o colores.

En el tercer capítulo se entrará en detalle en el método directo, que ignora todo lo anterior al rastrear la intensidad de los píxeles en lugar de formas en la imagen.

Según el tipo de cámara

En SLAM 3D tendremos siempre un input de imagen, casi siempre en color. Utilizando esta información se sigue pudiendo afrontar el problema de distintas formas.

Se conoce como SLAM monocular a aquel que utiliza exclusivamente la información RGB para todo el proceso. La simplicidad y precio de su hardware lo hacen ideal en aplicaciones donde la robustez no es una prioridad, aunque la complejidad de su implementación será mucho más compleja. Tradicionalmente era imprescindible recurrir a un Filtro Extendido de Kalman para rastrear los puntos característicos que se extraían, de manera similar a las predicciones de profundidad de un sistema 2D. En publicaciones muy recientes, sin embargo, se está empezando a popularizar el uso de múltiples frames consecutivos para triangular un punto y predecir su profundidad. La mayor contrapartida es el error que acumulan sus estimaciones, además de que se procesan múltiples frames consecutivos que aportan poca información nueva.

En el extremo opuesto estaría el uso de cámaras estéreo. De manera similar a como enfocan los ojos humanos, son capaces de calcular una distancia a los objetos basada en imágenes. Por esto mismo permiten aplicaciones comerciales donde interesa sustituir al humano como el coche autónomo o drones.

Una última combinación de sensores, menos habitual, es la utilización de una única cámara apoyada por un sensor de inercia. De esta manera se tiene una estimación de los giros y la incertidumbre de los puntos característicos se reduce drásticamente.

Online/Offline

En SLAM online, es decir, aquel que opera en tiempo real, se calcula principalmente la pose del punto de vista en cada momento. El cálculo de la trayectoria no es algo que interese sobre la marcha, ya que el objetivo es deducir como va cambiando la pose de la cámara.

Sin embargo, en las aplicaciones en las que interesa grabar una secuencia y analizarla a posteriori para ver si es óptima (offline) suele ser el objetivo principal obtener como producto final la trayectoria del robot. La gran ventaja de diseñar un sistema offline es la libertad a la hora de implementar recursos que lo hagan más potente, ya que un mayor tiempo de ejecución no es problemático.

2.2. Estado del arte y limitaciones

A día de hoy, con la gran variedad de soluciones existentes para el problema de SLAM, los investigadores normalmente enfocan sus esfuerzos en mejorar el algoritmo en casos de uso concretos. La práctica totalidad de publicaciones relacionadas en los últimos dos o tres años trata los temas de mejora en el reconocimiento de objetos, optimización del coste computacional o detección de personas y otros objetos móviles.

Aunque parece haber consenso en que la aplicación del EKF para reconocimiento y asociación de puntos es la solución más equilibrada en coste computacional y precisión, se han realizado pruebas implementando control robusto con resultado satisfactorio. La publicación [24] es una de las más interesantes en lo que llevamos de 2017 y en ella se recurre a redes neuronales convolucionales (CNN) para predecir la profundidad sin necesidad de un sensor ad hoc. La estrategia que se aplica empieza por hacer una predicción de alta incertidumbre y según se van obteniendo imágenes ir haciendo un mapa denso que permita refinar las predicciones sucesivas. Para ello se implementa un algoritmo *backpropagation* en el que se entrena una matriz que se opera con la salida de capas previas según el operador convolución (⊗). Según las pruebas realizadas por sus autores, iguala en rendimiento a cualquier sistema monocular avanzado (es decir sin input de profundidad) de los existentes y se comporta de manera excelente en entornos donde resulte difícil crear puntos característicos, como lugares sin muebles ni objetos que se puedan tomar como referencia. Esta ventaja tiene una contrapartida intrínseca, que radica en la pobre capacidad de reconocer lugares ya mapeados.

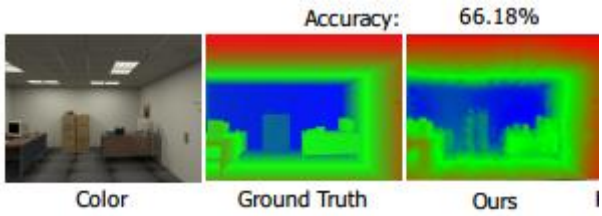


Figura 6: Resultado de la predicción de profundidad con CNN

Otras técnicas de control avanzado como el algoritmo DE (Evolución Diferencial) han sido aplicadas sobre todo a SLAM 2D y es difícil encontrar publicaciones donde se aplique a 3D. Esto se debe a un mucho mayor coste computacional por duplicarse el número de grados de libertad que se manejan. Sobre el papel sería posible que superara en rendimiento a muchas de las publicaciones de las que se está hablando pero los requisitos de sistema limitan la libertad a la hora de aplicarlo.

RGBDTAM y Elasticfusion son dos publicaciones referencia entre las recientes. La primera enfoca el problema desde la búsqueda de eficiencia mientras que la segunda presenta mayor complejidad y precisión.

RGBDTAM [32] es un sistema desarrollado por varios investigadores de la Universidad de Zaragoza utilizando como base trabajos previos y aportando una serie de innovaciones sobre

ellos, siendo el principal [25] en el que se combina por primera vez el cálculo de la odometría visual minimizando el error fotométrico entre frames con una predicción de profundidad deducida de reproyectar varios frames. Aporta algoritmos internos que descartan objetos móviles para no tomarlos como referencia y detección de cambios globales en la iluminación.

ElasticFusion [33] es uno de los trabajos más recientes y el más preciso dentro de RGB-D SLAM. Da mucha más importancia al mapeo de alta precisión que a la estimación de la trayectoria. Para ello ejecuta con éxito una estimación de la fuente de luz que descarta sombras y crea un mapa homogéneo.

ORB SLAM 2 [31] es otro sistema RGB-D reciente y avanzado. Su objetivo es operar en tiempo real en CPU, aun incorporando todos los subsistemas que cabe esperar de un SW puntero como cierre de bucles, optimización de reuso del mapa o relocalización. Siendo muy similar a los dos anteriores, su aportación con respecto a ellos es el tipo de algoritmo que deduce la trayectoria, basado en aplicar mínimos cuadrados para calcular la reproyección 2D de los puntos 3D en lugar del más habitual método fotométrico que se verá más adelante.

Otra de las publicaciones más recientes que aporta una innovación es Pinpoint SLAM de Mitsubishi [25]. Es un sistema RGB-D SLAM, en el que la mejora la encontramos en el aprovechamiento de información. Realiza SLAM 3D y 2D en paralelo con la misma secuencia de imágenes, reciclando para el algoritmo 2D la información que no aporta nada al proceso 3D. La mayor complicación y su clave de éxito es la búsqueda de correspondencia entre ambos algoritmos para minimizar el error.

De estos ejemplos y del apartado anterior se puede concluir que la investigación relacionada a este asunto está en un estado muy avanzado, donde existen soluciones adaptadas a cada tipo de aplicación y para cada tipo de sensor. Los límites están por lo tanto en las capacidades de computación de los equipos que ejecutan los programas. Aun así es cierto que en aplicaciones con cámara hay todavía una barrera importante en el filtrado de objetos dinámicos, lo que limita su rendimiento en espacios abiertos o donde trabajen personas u otros robots.

2.3. Aplicaciones habituales y potenciales

Aunque desde su creación SLAM ha sido muy utilizado en robots móviles para agilizar y automatizar tareas industriales, en la última década se han multiplicado las áreas de aplicación de esta tecnología gracias a la evolución del hardware y a la aparición en el mercado de nuevas necesidades, tanto industriales como domésticas.

Mapeo 2D

La variante más sencilla de SLAM, utilizando mapeo 2D, es suficiente para aplicaciones en habitáculos planos donde la información de distancia es la única fuente necesaria. Es una aplicación relativamente primitiva que resulta útil en robots domésticos como aspiradoras autónomas o en líneas de montaje en las que se utilizan carros automatizados para el transporte de piezas.

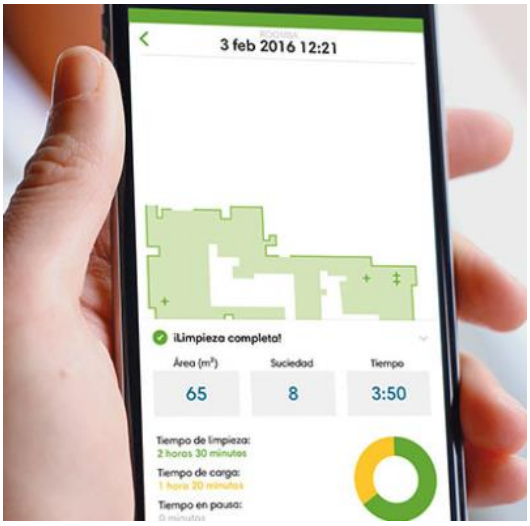


Figura 7: Aplicación móvil de iRobot donde se visualiza el plano creado

Mapeo como alternativa a otros métodos de localización

Siempre que hablamos de mapeo y localización mediante SLAM vienen a la mente aplicaciones en lugares pequeños y cerrados como fábricas y viviendas, donde una implementación con GPS sería imprecisa y costosa. Sin embargo, el potencial de SLAM en lugares amplios es considerable si se cuenta con los sensores adecuados. Por lo tanto es posible automatizar tareas allí donde el GPS no es una opción, como en exploraciones del fondo marino, minas, túneles e incluso en otros planetas. Su implantación tanto en robots como en drones para este tipo de tareas es ya una realidad.

Por otra parte, y de manera similar, hay variedad experimentos en los que se aplica SLAM al coche autónomo. Los sensores utilizados poco se parecen a aquellos que utilizaremos en aplicaciones de investigación en espacios reducidos, siendo necesarias varias cámaras, sensores LIDAR o similares y el apoyo de GPS y odometría. Aun así, se sigue el mismo principio de funcionamiento por el cual se desea crear un mapa y navegar por él de forma cada vez más precisa.

Realidad aumentada

El trabajo de Klein y Murray en 2007 [1] es una de las publicaciones fundacionales en lo que a realidad aumentada aplicada al entretenimiento se refiere. Vemos como se utiliza la información de profundidad para ubicar elementos digitales, complementada con un algoritmo SLAM para que el punto de vista mantenga coherencia con dichos elementos. Este principio ha evolucionado en la década que ha pasado desde entonces hasta formar parte de

aplicaciones comerciales completamente operativas, sobre todo en el ámbito de los videojuegos. En estos casos se sustituye al robot por el usuario humano, ubicándose los sensores RGB y de distancia en unas gafas que a la vez hacen de pantalla donde se proyectan los elementos añadidos por el software.

Figura 8: Captura del sistema de Klein y Murray

Este tipo de aplicación está empezando a ser utilizado por empresas de tecnología médica, aeroespaciales y automovilísticas para formar a sus trabajadores en escenarios semivirtuales ya que puede suponer un importante ahorro en costes y sobre todo en riesgo.

Figura 9: RA aplicada a formación de mecánicos

Por su parte Google ha hecho su incursión en el mundo de la realidad aumentada desde otro enfoque con su proyecto Tango. Tango es una plataforma para desarrolladores de aplicaciones móviles, por lo que parte de la difícil premisa de que el Smartphone cuenta con sensor de profundidad o cámara estéreo. Está eminentemente enfocado al área de entretenimiento, siendo una de sus aplicaciones más interesantes la previsualización de artículos en compras online, como muebles. Se encuentra en fase beta por lo que apenas

existen fabricantes de teléfonos involucrados, de manera que la mayor barrera para su llegada al público será el coste de estos terminales.

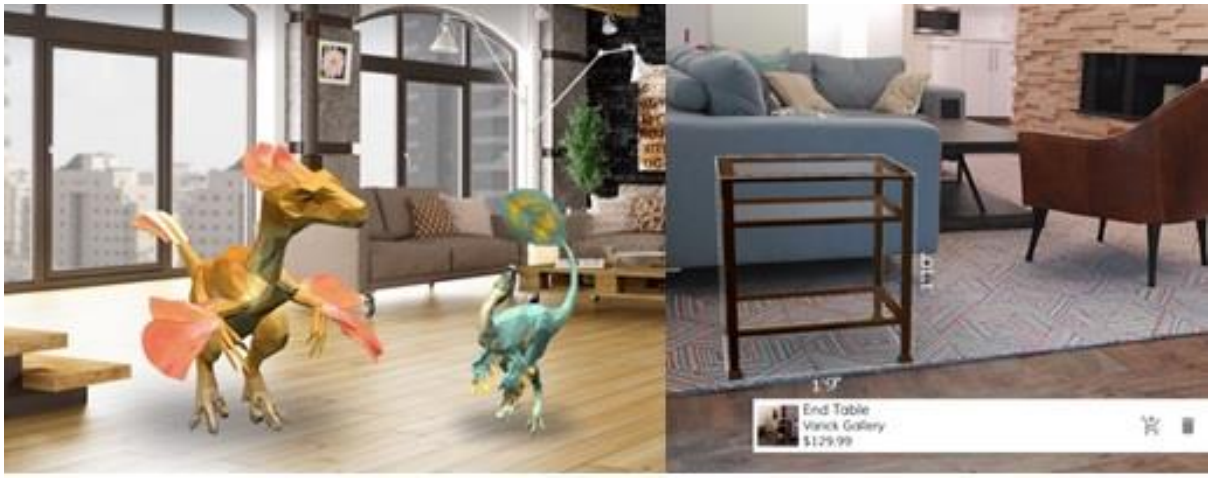


Figura 10: Capturas de Google Tango

Aplicaciones en medicina

Existen multitud de intervenciones quirúrgicas que pueden beneficiarse del SLAM. En particular aquellas en las que una cámara examina el interior de una cavidad como la laparoscopia. En estos casos la inclusión de SLAM permite mapear las cavidades a examinar y estudiarlas posteriormente, reduciendo el tiempo de incursión en el paciente. A nivel académico, estos mapas se pueden combinar con realidad aumentada para etiquetar partes del cuerpo, realizar medidas o simplemente generar reconstrucciones realistas. Este es un primer paso fundamental de cara a la robotización de tareas de este tipo.

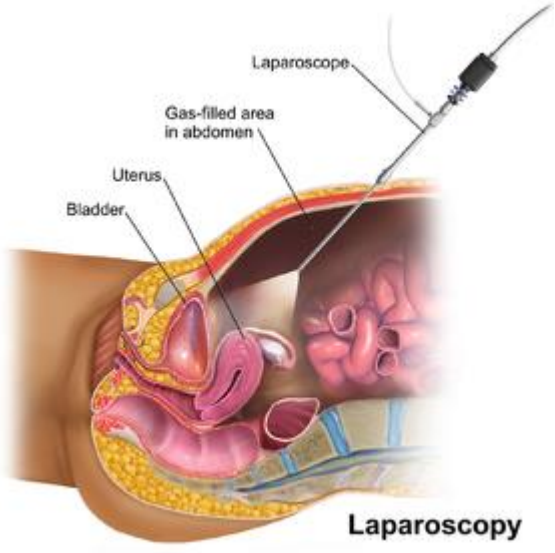


Figura 11: Proceso de laparoscopia

Escaneo de objetos en 3D

En los últimos tiempos se ha popularizado la práctica de generar archivos de objetos 3D a partir de escaneos. Esto tiene diversas aplicaciones posibles en el mundo de la realidad aumentada, ya que permite introducir en el universo virtual objetos escaneados previamente por el usuario, tales como poner su cara en un personaje de videojuegos. Una técnica similar se ha visto utilizada para incluir edificios reales en mapas virtuales como Google Maps o en efectos especiales de vídeo.

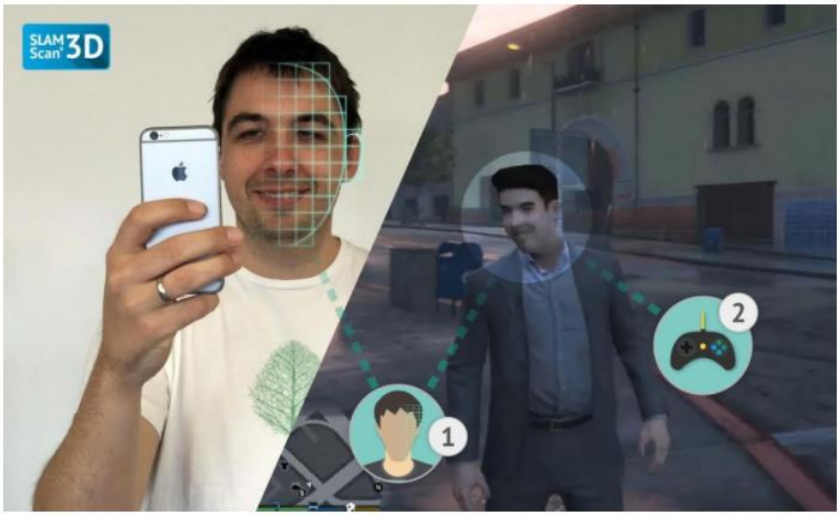


Figura 12: Imagen promocional de Dacuda, empresa de videojuegos

.

CAPÍTULO 3. Caso de estudio: RGB-D SLAM y su implementación

Una vez introducidas las características y tipos de problemas a los que nos podemos enfrentar, se va a desarrollar en profundidad el caso de estudio elegido.

El motivo de elegir RGB-D SLAM tiene que ver con el hecho de que es el más explotado en investigación en los últimos años y por lo tanto el que más aplicaciones modernas tiene. Su interés radica en las diversas maneras de implementarlo según las necesidades que se quieran cubrir, como ya se ha ido viendo en los capítulos previos. ++

En esta sección se describirá en detalle la arquitectura típica de un sistema de este tipo, explicando cada uno de los subsistemas y su aportación al conjunto.

Para implementar un sistema RGB-D, será necesario además de un sensor como la Kinect, una plataforma sobre la que desarrollar SW como MATLAB o un entorno de programación en C++, y un equipo en el que correr un SW de este nivel de peso. Lo habitual es trabajar en Ubuntu por su facilidad para trabajar con librerías de terceros y su bajo consumo. Otro motivo es que además, para acceder a los recursos del sistema ROS, que se explica en el capítulo 4, es necesario un OS basado en Linux.

Como se ha podido entender en secciones previas sobre las aplicaciones y diseño de la tecnología RGB-D, para resolver el problema contamos con datos de profundidad y de color de cada píxel de un determinado frame. Con estos datos tenemos dos caminos, el tradicional y el directo.

El primero consiste en extraer características que puedan ser usadas de referencia de la imagen a color en 2D, y ubicarlos en un mapa 3D gracias a la coordenada z que nos da el sensor de profundidad. Hecho esto se implementa el método probabilístico que se desee, como el filtro de Kalman, para a cada lectura predecir si un punto característico observado es el mismo que en el frame anterior. Si hay respuesta positiva, se traza la línea y se anota el desplazamiento.

Un método mucho más moderno es el llamado método directo o de odometría visual. En él se recurre identificar la matriz que transforma un frame en otro a partir de reproyecciones.

En cualquiera de los dos casos, las funciones que realizan los sistemas RGB-D se pueden categorizar según los siguientes subapartados.

3.1. Arquitectura del sistema

A alto nivel, las tareas que ha de llevar a cabo un software RGB-D SLAM son principalmente cuatro: reconocer puntos de referencia, usarlos para calcular el desplazamiento, crear el mapa y reconocer zonas donde ya se ha estado.

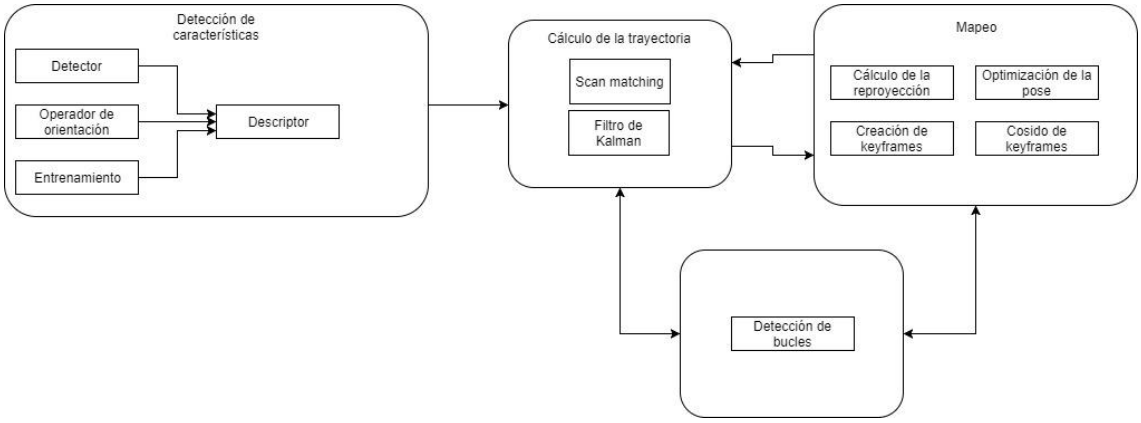


Figura 13: Arquitectura típica de RGB-D SLAM

Los enfoques más eficientes reparten estas fases en hilos en paralelo. Debido a la complejidad de las tareas, realizarlas de forma secuencial frame a frame simplemente hace imposible la operación en tiempo real. También es habitual que la creación de mapas se realice en la GPU en lugar de en la CPU, liberando a esta última de esa carga para agilizar los cálculos de la trayectoria. De no hacerse así, los métodos de selección de características pueden resultar demasiado pesados para permitir operar en tiempo real.

3.2. Extracción de puntos característicos

En los sistemas SLAM basados en puntos característicos suele ser habitual que haya que identificarlos sobre la marcha. Esto supondrá complicaciones por la complejidad creciente, de ahí la importancia de reconocer estos puntos una vez creados. Para reducir el coste computacional, las características extraidas de los frames deben ser fáciles de identificar y no demasiado numerosas. Por ello los enfoques que se toman para afrontar este problema incluyen siempre una aproximación que se calcula mediante detectores y/o descriptores. Un detector simplemente es un algoritmo que detecta formas, texturas o colores en una imagen, devolviendo unas coordenadas 2D. Idealmente, un detector se complementa con un descriptor, que además analiza la zona vecina al punto detectado con la finalidad, de optimizar el muestreo o de reconocer la orientación del punto.

Un buen ejemplo es el caso del método SURF (Speeded-Up Robust Features) se recurre a un filtro de caja, que no es más que una versión simplificada del filtro gaussiano. Se localizan píxeles de alto gradiente y se asigna su valor a los vecinos de intensidad similar. FAST es un método similar que compara cada píxel con una matriz de vecinos del más lejano al más cercano para decidir si es o no un punto de interés.

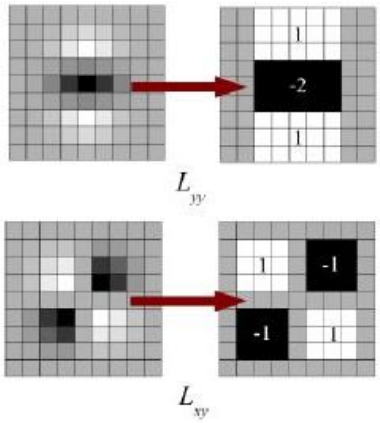


Figura 14: Funcionamiento del detector SURF

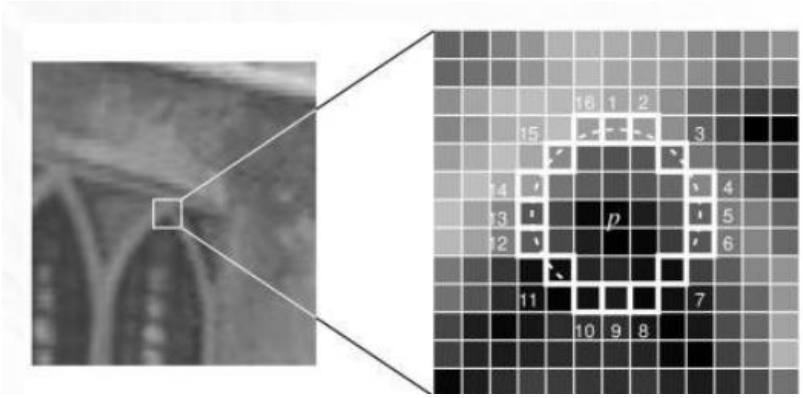


Figura 15: Funcionamiento del detector FAST

ORB (Oriented FAST and Rotated BRIEF) es una de las publicaciones más avanzadas en este área por su manejo de los recursos de la CPU, siendo rápido y eficaz además de tolerar bien el ruido. Es por ello uno de los sistemas más implementados en los trabajos de investigación más recientes. En ORB se combina el detector FAST con el descriptor BRIEF [34].

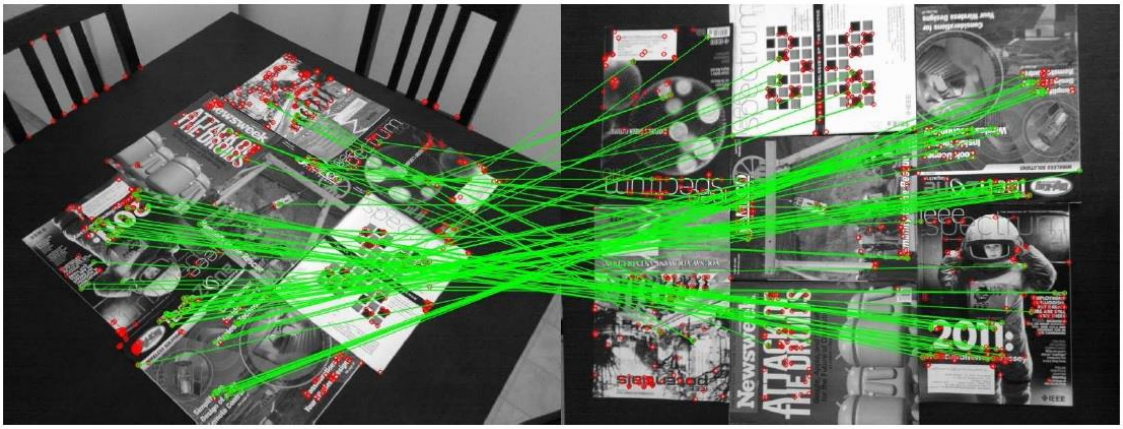


Figura 16: Resultados de ORB en imagen sin filtrar

ORB no es un descriptor como tal sino que puede considerarse como un paquete de recursos de obtención y rastreo de puntos de referencia. Como se puede ver en la figura 16 además de

crear dichos puntos, es capaz de reconocerlos en otra imagen gracias a su proceso de aprendizaje.

Otra forma de afrontar el problema es detectar bordes en lugar de puntos. Los algoritmos de Canny, Sobel, Harris o Prewitt de manera similar aplican una reducción gaussiana, buscando la dirección del gradiente de cada píxel y de esta forma filtrar las zonas sin textura. La figura 18 es un gran ejemplo de por qué la medida de profundidad puede ser muy útil.

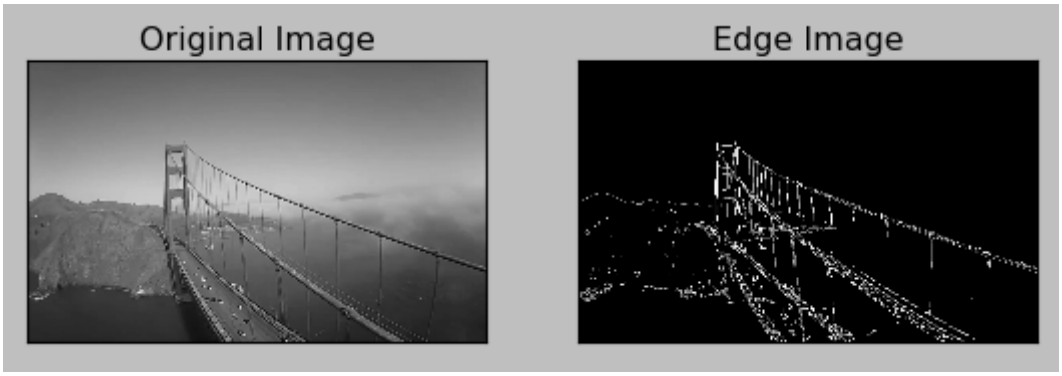


Figura 17: Resultados de la detección de bordes

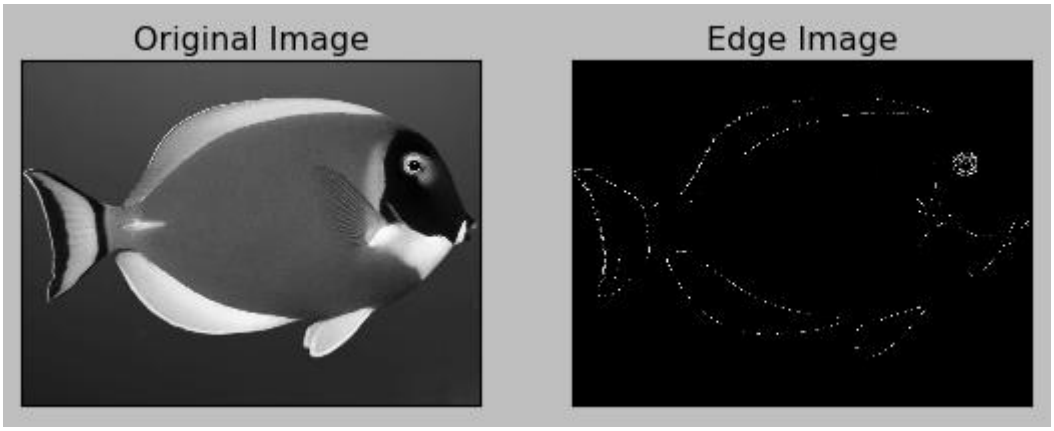


Figura 18: Resultados de detección de bordes

3.3. Cálculo de trayectorias

Como se introdujo previamente, para deducir el desplazamiento de punto de vista a partir de imágenes se ha de encontrar una manera de identificar las mismas características en frames sucesivos. A continuación se detallan las dos formas más avanzadas de resolver el problema.

Scan Matching

La manera más popular de resolver este problema es mediante Scan Matching (emparejamiento de escaneos) y consistirá en encontrar una transformación rígida de traslación y rotación que alinee dos frames sucesivos. Otros sistemas con mayores exigencias han implementado con éxito emparejamientos entre mapas completos o entre una imagen y un mapa, pero son soluciones desproporcionadamente costosas computacionalmente hablando para el tipo de casos que queremos resolver.

Para analizar dichos frames se toman como datos de entrada los puntos característicos que estén a la vista en ambos frames. Sean por lo tanto dos sets de puntos:

$$X = \{x_1, ..., x_n\}$$
$$P = \{p_1, ..., p_n\}$$

Expresión 1: Nomenclatura de los sets de puntos

Una manera popular de proceder es aplicar mínimos cuadrados para buscar la línea que conecte los frames. En este caso se quiere obtener la rotación R y la traslación t tal que se minimice la expresión de mínimos cuadrados:

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$$

Expresión 2: Reducción por mínimos cuadrados típica

Donde xi y pi son el mismo punto en ambos frames.

Este método es sencillo de implementar pero es muy propenso a errores de precisión, debido a que toma en cuenta puntos que pueden no ser relevantes para la transformación. Esto puede influir en el resultado de manera radical si se detectan demasiadas falsas referencias.

Para solucionar esto existen diversas formas de filtrar los puntos que no deseamos tener en cuenta. La más popular es el algoritmo RANSAC (RANdom SAmple Consensus)

RANSAC es un método iterativo para calcular los parámetros de un modelo matemático de un conjunto de datos observados que contiene valores atípicos. Es un algoritmo no determinista en el sentido de que produce un resultado razonable sólo con una cierta probabilidad, mayor a medida que se permiten más iteraciones. El algoritmo fue publicado por primera vez por Fischler y Bolles SRI International en 1981.

Puesto que los valores atípicos que queremos ignorar suelen venir de picos de ruido o falsas detecciones, es necesario lo primero establecer un criterio por el que filtrar estos valores.

La manera de operar de RANSAC es muy intuitiva y se resume en los siguientes pasos.

- Dado un paquete de puntos, el programa toma una muestra pequeña y crea el modelo. Puede ser en base a mínimos cuadrados o a cualquier otra técnica más compleja que se desee implementar.
- Se habrá obtenido ahora una posible línea como solución, pero se ha de comprobar si se ajusta al resto de puntos.
- Se mira cuántos y cuáles son los puntos que encajan. Los parámetros que determinan cuántos puntos se deben ajustar para que resulte válido los ajusta el usuario, así como el criterio o la tolerancia por los cuales un punto se ajusta al modelo.

Si tras crear el modelo se recuenta que no se ajusta a un suficiente porcentaje de puntos, simplemente se repite con otra muestra.

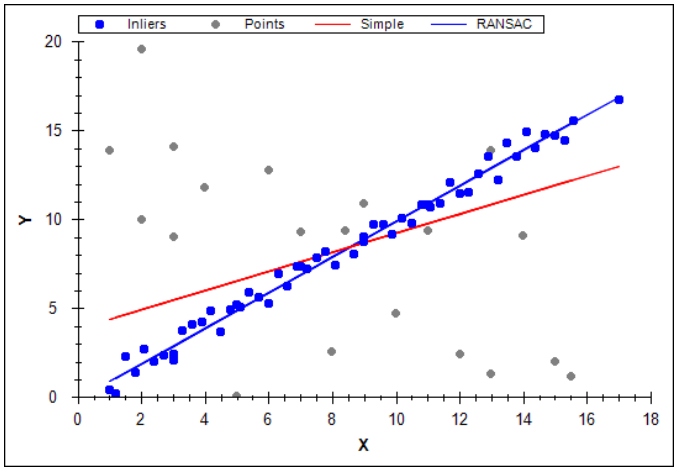


Figura 19: Comparativa de mínimos cuadrados contra RANSAC

En la anterior figura podemos apreciar de un vistazo la enorme influencia que puede tener incluir en el modelo los puntos atípicos. Es por lo tanto fundamental para deducir una trayectoria fiable incluir alguna estrategia de este tipo cuando se trabaja con sensores propensos a error como las cámaras.

Filtro de Kalman

El filtro de Kalman es una herramienta probabilística perfeccionada a lo largo de 60 años de historia, y por tanto la más adoptada en aplicaciones tecnológicas que incluyan navegación, control o guiado.

Su finalidad en SLAM es, dada una observación y una acción de movimiento, proporcionar la nueva posición de los puntos de referencia y la trayectoria que se ha seguido. Dicho de otra manera, si definimos el estado del robot como su posición respecto de los puntos característicos, iremos obteniendo los estados sucesivos.

Para ello se sigue una estrategia de predicción más corrección. Esto es, cuando observamos un punto característico queremos estar seguros de no estar confundiéndolo con otro, por lo que se hace una predicción y esta incertidumbre es la que se corrige. Una de sus ventajas radica en esta naturaleza recursiva, es decir, en cada iteración utiliza información del estado anterior y no se necesita almacenar información que limite la toma de nuevos datos.

En resumen, el algoritmo sigue 5 pasos:

- Predicción del estado
- Predicción de la medida
- Observar la medida
- Asociar los datos
- Actualizar la predicción con la observación.

Se ha dicho que como input es necesaria la acción de movimiento que se realiza sobre el robot. En casos de robots con sensores de desplazamiento o controles externos este dato puede resultar trivial, mientras que en SLAM lo que necesitamos es la transformación que resulte del scan matching, que en definitiva es la trayectoria que hemos deducido que realiza el observador.

El filtro de Kalman asume dos situaciones. En primer lugar se da por hecho que el entorno no cambia a lo largo de la operación, es decir, no transitan otras personas o robots por la zona de observación ni el robot mueve los objetos en caso de impacto. En segundo lugar, se necesitan modelos de transición y observación lineales (x y z respectivamente), de la forma:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

$$z_t = C_t x_t + \delta_t$$

Expresión 3: modelos lineales de transición y observación

Donde A_t (n x n) describe la transición hasta el estado actual x_t desde x_{t-1}.

B_t (n x l) es el cambio provocado por el input de control u_t.

C_t (k x n) describe cómo es el mapa en x_t para la observación z_t.

ϵ y δ son parámetros estimados en relación al ruido.

Se necesita que los modelos sean lineales porque de esa manera se pueden tratar las predicciones y las observaciones como distribuciones gaussianas.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\Big)$$

Expresión 4: Ecuación de la gaussiana

1: **Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$
3: $\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

4: $K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$
6: $\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$
7: *return* μ_t, Σ_t

Figura 20: Algoritmo del filtro de Kalman

En el algoritmo de la figura 20 μ_t es el estado y z_t la observación. Vemos que aparece la matriz de covarianzas Σ, que juega un papel fundamental pues incluye información sobre la incertidumbre. K_t es un parámetro que tiene en cuenta las incertidumbres y las observaciones para cuantificar cómo de seguro está el robot de sus conclusiones. Cuanto mayor es K menor es la incertidumbre.

Cuando hablamos del paso de corrección, el cuarto de los cinco enunciados anteriormente, hacemos referencia a una media ponderada entre las curvas de predicción y observación. Para calcular el peso de cada una en la media se tiene en cuenta la incertidumbre de la predicción pero también la fiabilidad del sensor. En la figura 21 podemos ver el resultado. Resulta intuitivo ver que la diferencia entre observación y predicción se acercará cada vez más si se repiten observaciones. Esto es, recorrer zonas ya mapeadas reduce la incertidumbre asociada a los puntos característicos en ellas.

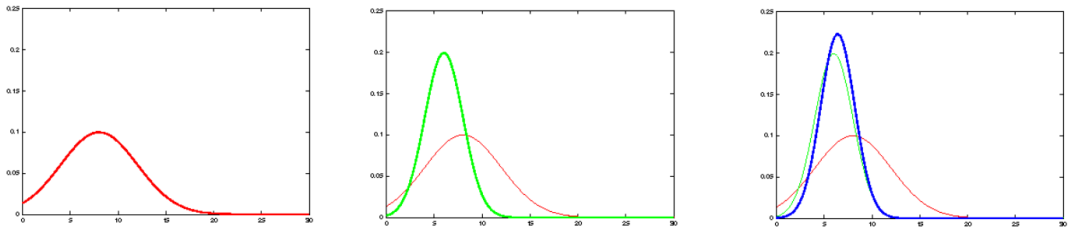


Figura 21: Curvas de predicción, observación y corrección

La complejidad del proceso, y por lo tanto el factor que limitará el tiempo de aplicación, viene del tamaño de la observación. Un mayor número de puntos de referencia influye directamente en la dimensión de la matriz de covarianzas Σ y por tanto en los vectores de observación y estado (figura 22). Es por tanto necesario establecer una cantidad limitada de referencias mediante otros métodos externos. Lo más sencillo es trabajar con referencias creadas a mano, aunque los métodos más avanzados admiten dimensiones dinámicas.

$$\mu_0 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \end{pmatrix}^T$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{pmatrix}$$

Figura 22: Vector de estados y matriz de covarianzas en la inicialización

Por esto mismo uno de los grandes problemas a resolver es la alta incertidumbre en el momento de la inicialización. Si en t=0 no hay observaciones que incluyan puntos característicos se sigue la navegación de manera libre o controlada hasta que las haya y entonces se puedan empezar a reducir las incertidumbres.

Por otro lado, el problema al que nos enfrentamos es no lineal. Por esto se efectúa una linealización alrededor del punto en cuestión y se sigue el algoritmo estándar (figura 20). Para hacer la linealización se hace una expansión de Taylor aplicando la jacobiana. Esto se conoce como Filtro Extendido de Kalman (EKF).

El modelo linealizado se aplica al algoritmo anterior sustituyendo las matrices de transición A y observación C por las respectivas Jacobianas de cada modelo. Queda entonces el algoritmo de la figura 23, donde g y h son las funciones que modelan transición y observación. En la práctica se demuestra que con una linealización precisa este método funciona de manera adecuada con no linealidades moderadas. El hecho de que sea un método aproximado significa que cuanto mayor es la incertidumbre, más se ve afectado el error que se comete.

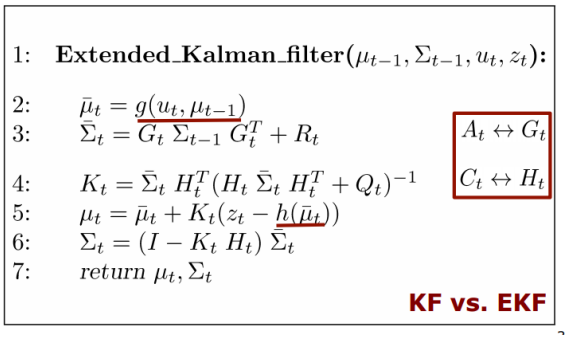


Figura 23: Algoritmo EKF

Alternativas: Método directo

El método directo es una aplicación dentro del campo de la tecnología conocida como Structure from Motion (SfM) que hace referencia a todo tipo de problema en que se deducen cualidades del entorno a partir del movimiento del punto de vista.

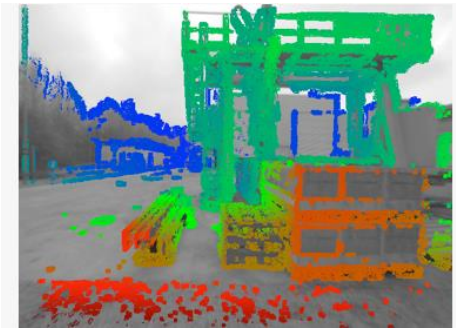
Es una estrategia moderna cuya mayor ventaja es la posibilidad de crear un mapa denso aprovechando mucha más información de la imagen en lugar de unos pocos puntos característicos. Para ello aplica técnicas de extracción de puntos característicos como las vistas en 3.2 donde a partir de la imagen, generalmente reducida a escala de grises, rastrea similitudes entre frames en base a su intensidad entre blanco y negro. Los llamados píxeles de alto gradiente son los más cercanos a los extremos y por ello mismo son fáciles de rastrear.

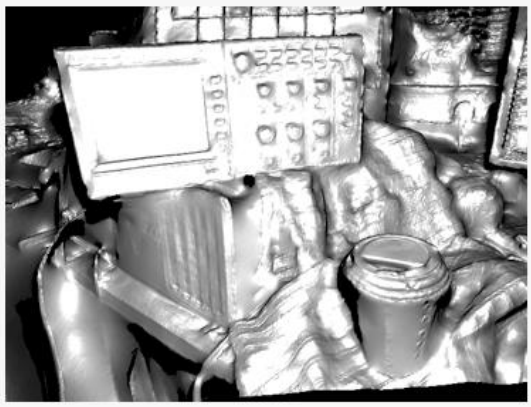
Hay varias posibles técnicas a la hora de deducir la transformación entre frames, partiendo de la premisa de que a cada punto se le debe poder aplicar el mismo criterio de transformación.

La técnica a emplear en este caso consiste en tratar la imagen como una matriz y minimizar el error de fotometría que suele tomar una forma similar a la expresión 5.

$$E(\boldsymbol{\xi}) = \sum_i \left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \boldsymbol{\xi}))\right)^2,$$

Expresión 5: Minimización por Gauss-Newton





Las formas de minimizar el error pueden ser varias. La más popular por su equilibrio entre complejidad y resultado es la de Gauss-Newton (expresión 6). Existen otros enfoques habituales como el de Huber o simplemente aplicar mínimos cuadrados.

$$E_{\mathbf{p}j} := \sum_{\mathbf{p}\in\mathcal{N}_\mathbf{p}} w_\mathbf{p} \left\| (I_j[\mathbf{p}'] - b_j) - \frac{t_j e^{a_j}}{t_i e^{a_i}} (I_i[\mathbf{p}] - b_i) \right\|_\gamma$$

Expresión 6: Optimización de Huber

No se entra en detalle sobre cada fórmula porque cada modelo utiliza una serie de parámetros para definir las imágenes y las transformaciones, pero se puede observar que ambas buscan la diferencia entre una imagen I y una imagen transformada I'.

Para poder aplicar la expresión del error hace falta obtener una transformación rígida en 3D que puede ser de la forma:

$$\mathbf{G} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$$

donde los elementos de la primera fila son la rotación y la traslación que experimenta el punto de vista.

Todo este proceso no se realiza entre frame y frame sino que para no arrastrar errores se compara el frame actual con un frame de referencia o *keyframe.* Una vez el keyframe está demasiado lejos del actual para poderse hacer una transformación fiable se elige uno nuevo y se reinicia el proceso.

Este método, si bien consume por lo general más recursos y sus algoritmos son más complicados que el método tradicional, permite ampliar el rango de aplicaciones gracias a que puede manejar con soltura imágenes con cientos de puntos característicos.

3.4. Mapeo

La idea de la creación de un mapa a partir de la información de la cámara se puede reducir a un problema de minimizar una función no lineal y representarla en un gráfico. En resumen lo que se desea es elegir una serie de puntos (o todos los posibles según la exhaustividad del método aplicado) y crear un mapa 3D con esa nube de puntos. La clave del problema es identificar esos puntos cuando están sometidos a una incertidumbre causada por ruido gaussiano.

A nivel general, lo primero será establecer una correspondencia entre varias imágenes, en base a unas características. Hecho esto es necesaria una manera de identificar el mismo punto en distintos frames. Hasta aquí puede verse que se sigue un enfoque prácticamente idéntico al cálculo de trayectoria. Resuelta esta parte esto el algoritmo sigue como en la figura 24.

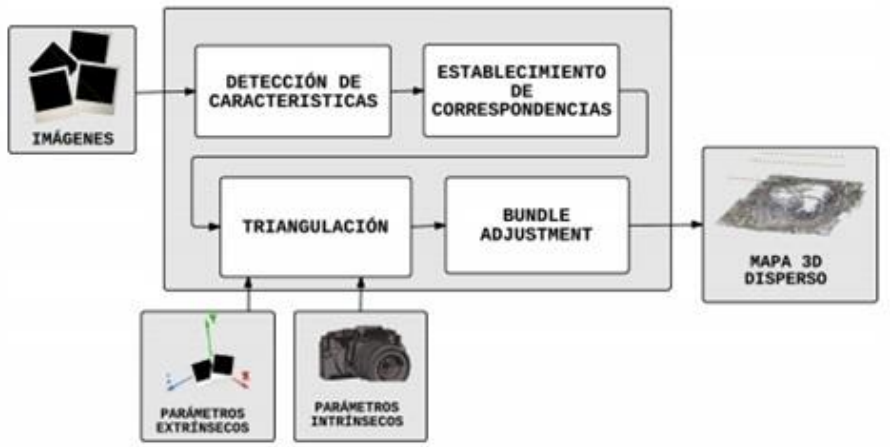


Figura 24: Algoritmo de mapeo

Se conoce como triangulación al proceso de determinar la posición del punto P en el espacio físico, a partir de al menos dos imágenes distintas en las que se pueda ver el punto, y el conocimiento de la pose de las cámaras para cada una de las imágenes. Por otra parte, el proceso de convertir datos 3D en imágenes planas se denomina reproyección.

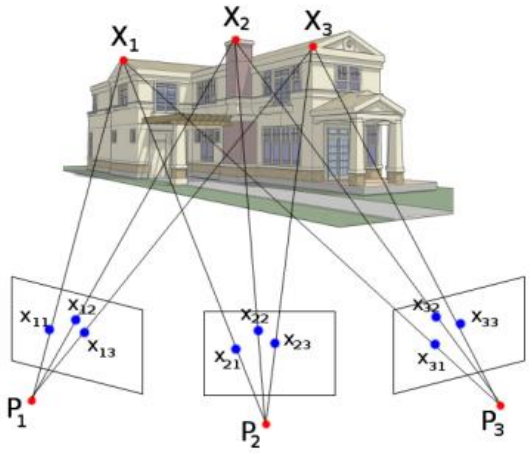


Figura 25: Triangulación de puntos en varios frames

Para optimizar este proceso se recurre a algoritmos como Bundle Adjustment. Sean n puntos 3D vistos desde M vistas diferentes, x R ij la re-proyección del punto i en la imagen j y xij el punto original. Sea vij una variable binaria igual a 1 si el punto i es visible en la imagen j y 0 en caso contrario. Se define bundle adjustment como el proceso que minimiza el error de reproyección total según la expresión 7:

$$\min_{a_j, b_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} \left\| x_{ij} - x_{ij}^{R} \right\|$$

Expresión 7: Mínimos cuadrados para BA

Para conseguir reducir el error de reproyección se utiliza un algoritmo de minimización no lineal robusto que modifica los valores estimados de los parámetros conjuntamente, o dicho de otra forma, intenta ajustar el grupo (bundle) de rayos que unen las proyecciones y los puntos físicos de tal forma que el error de reproyección sea el menor posible. Este método tiene más de 60 años de historia pero ha incrementado su uso en los últimos años gracias a los avances en visión por computador.

Al tratarse de un problema de optimización sobre una función no lineal, bundle adjustment hace uso de algoritmos de mínimos cuadrados. Uno de los más comunes es Levenberg-Marquardt que ha probado ser uno de las mejores opciones por su fácil implementación y rápida convergencia. Levenberg-Marquardt necesita un punto de partida, un conjunto de valores para los parámetros y las mediciones de error para dichos parámetros. A cada iteración de LM se calcula un nuevo conjunto de parámetros que reduce el error, hasta llegar

a la solución óptima, y la modificación que se produce entre una iteración y la siguiente es aquella que minimiza el error en un cierto grado.

3.5. Detección de bucles

Un bloque fundamental en los sistemas modernos es la detección de bucles. Su función es reconocer áreas ya mapeadas para optimizar la navegación por zonas conocidas. Una vez identificada una zona ya mapeada, las incertidumbres se reducen drásticamente. Es por lo tanto fundamental seguir una estrategia robusta y conservadora en la detección de bucles, puesto que cerrar un bucle en falso supondría no sólo un error en la ubicación que cree el robot sino en la forma del mapa, haciéndose necesaria una reinicialización del sistema.

El principio que se sigue consiste en identificar cuándo la transformación entre el keyframe actual y el frame actual da como resultado un keyframe previo, lo que significaría que ya se ha visualizado antes.

Para ello se recurre al llamado vocabulario visual, que resulta de codificar los keyframes como vectores de características a mucha menor resolución para poder acceder a ellos y hacer las comparaciones necesarias.

Estos vectores de características contendrán información de intensidad de colores, distancia o puntos característicos. Los métodos más populares para codificar estas imágenes son:

- Cuadrícula regular

Este tipo de método es uno de los más simples y eficaces que podemos utilizar a la hora de detectar características. La imagen se divide en partes iguales y estas partes son los puntos de interés. La única limitación que hay en este método es que utiliza poca información de la imagen.

- Detector de los puntos de interés

Este tipo de detectores marcan como puntos de interés las manchas, bordes o esquinas de la imagen. Considera que estos son los puntos más importantes porque son los primeros que detecta el ojo humano.

- Hessiano/Laplace

Es un detector basado en la matriz Hessiana. Este detector aplica el determinante para elegir la posición y la escala. Por lo tanto, dado un punto p = (x, y) de la imagen I, la matriz Hessiana H (p, u) se define como:

$$\mathbb{H}(p, o) = \begin{bmatrix} L_{x,x}(p, o) & L_{x,y}(p, o) \\ L_{y,x}(p, o) & L_{y,y}(p, o) \end{bmatrix}$$

CAPÍTULO 4. RGBDTAM y Trabajo relacionado.

En este capítulo se dará una visión detallada del sistema que se ha decidido analizar dentro de las más recientes publicaciones de RGB-D SLAM.

RGBDTAM es un sistema lanzado a principios de 2017 por dos investigadores de la Universidad de Zaragoza. Por su vocación de investigación está disponible sin coste la colección de paquetes en su página de GitHub para todo aquel que desee hacer experimentos o aportar mejoras.

Se trata de un sistema basado en trabajos previos, de los mismos autores y de otros, que busca por encima de todo resolver el problema de la manera más eficiente, no siendo necesariamente el que mayor rendimiento entregue. Otra premisa que se sigue a conciencia es ofrecer un sistema polivalente, válido para cámaras con o sin sensor de profundidad, y válido para aplicaciones tanto online como offline.

Una de las principales aportaciones de RGBDTAM es la distribución de las funciones en hilos paralelos en la CPU, liberando de trabajo a la GPU que es precisamente el factor que limita la operación en tiempo real en equipos de gamas medias y bajas. La idea en sí no es nueva, y se conoce como PTAM (Parallel Tracking And Mapping), pero sí hay aportaciones en la manera de implementarla.

En general se sigue un planteamiento basado en el método directo pero con la adición de mejoras de cara a la eficiencia. En cuanto a la trayectoria, se obtiene por el método tradicional en RGB-D, calculando el error fotométrico con la información de color y el geométrico con la de profundidad pero filtrando la información mediante detección de bordes.

Por otro lado, para el mapeo, el error fotométrico se obtiene a partir de la reproyección sobre un plano de varios frames consecutivos en lugar de uno sólo, como se hace en los sistemas monoculares más avanzados. Así se obtiene una estimación precisa sin la necesidad de un mapa denso de color. Sin embargo además, al contar con la información de profundidad, se complementa lo anterior con una observación más fiable de la estructura, lo que permite la creación de un mapa denso de puntos.

4.1. Arquitectura PTAM

El sistema utiliza una variante de los sistemas introducidos en 3.1 sobre la distribución de funciones en hilos paralelos. Se basa en la arquitectura PTAM (Parallel Tracking and Mapping) desarrollada en [1]. Se persigue con este planteamiento conseguir la operación en tiempo real en CPU.

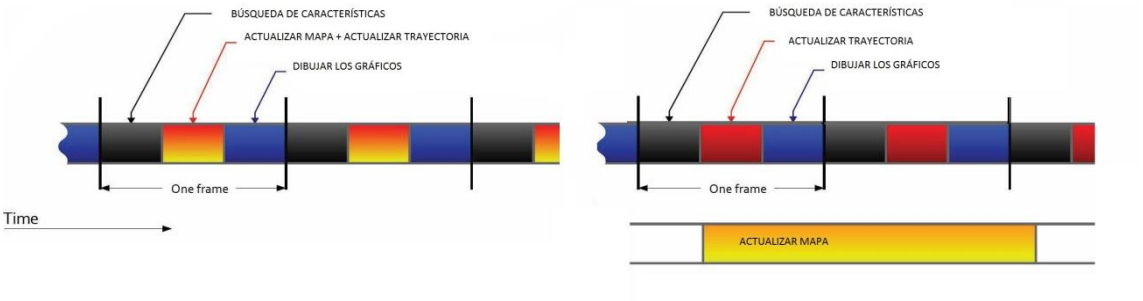


Figura 26: Esquema PTAM

El hilo de mapeo estima los mapas a partir de un set de keyframes {K1, . . . , Kj , . . . , Km}. Cada keyframe Kj = {Twj, Pj} se modela mediante su pose Twj donde w es el entorno y Pwj j w = {pw1, . . . , pwi, . . . , pwn} donde cada pwi contiene información fotométrica y geométrica. Por su parte el hilo de trayectoria estima la pose del frame actual minimizando errores de reproyección fotométricos y geométricos dentro de su nube de puntos alcanzados, en relación al keyframe actual. El proceso de detectar si nos movemos por zonas conocidas o nuevas que gestiona la creación o reúso de keyframes tiene lugar en este hilo también, al estar relacionado con la trayectoria. Estas tareas se enmarcan en la etapa de cierre de bucles que concluye el algoritmo.

4.2. Estimación de la posición

Detección de bordes mediante Canny

El algoritmo de John Canny [28] detecta bordes a partir de datos de profundidad, en base a la intensidad de los píxeles. Muchos sistemas SLAM utilizan la nube de puntos obtenida directamente para efectuar un scan matching sobre estos datos y obtener el desplazamiento. En particular en aquellos que no cuentan con el apoyo de una medida de distancia.

En SLAM directo no es habitual la implementación de un algoritmo de este tipo. Una de las ventajas del método directo es la posibilidad de utilizar el 100% de la información de la imagen para conseguir una deducción de la trayectoria robusta y un mapa denso detallado. Esta es la práctica típica, vista en 3.3.b, pero requiere una GPU de muy alto nivel para funcionar en tiempo real. En este SW sin embargo se filtra la imagen buscando bordes para calcular únicamente la traslación de estos.

El algoritmo que se sigue se podría resumir en 5 fases:

1. Aplicar filtro gaussiano para reducir el ruido

La aplicación de un filtro gaussiano a cada uno de los píxeles da como resultado una imagen de menor resolución, lo que de primeras ya es una buena noticia pues habrá que procesar menos información. El filtro aplica una distribución gaussiana a la intensidad del píxel de manera que los bordes finos se difuminan más cuanto más exigente sea el filtro hasta incluso desaparecer.

El filtro aplicado toma la forma del producto de dos gaussianas:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Expresión 8: Filtro gaussiano

Queda perfectamente claro al ver la figura 27 que cuanto más se filtra la imagen más sencilla es de tratar y más definidas quedan las principales formas presentes, perdiendo los detalles menores.

Figura 27: Resultado del filtro gaussiano para distintas ganancias

2. Localizar los puntos de alta intensidad

Una forma de modelar una imagen plana es:

$$f(x) = \frac{I_r - I_l}{2}\left(\operatorname{erf}\left(\frac{x}{\sqrt{2}\sigma}\right) + 1\right) + I_l.$$

Expresión 9: Modelo de imagen 2D

A partir de lo cual se pueden identificar bordes aplicando los límites:

$$I_l = \lim_{x \to -\infty} f(x) \; I_r = \lim_{x \to \infty} f(x)$$

Expresión 10: Límites de la intensidad del píxel

Siendo I_l la intensidad del píxel a su izquierda, I_r la intensidad a su derecha, y σ la ganancia que cuantifica el nivel de borrosidad.

3. Suprimir los no máximos

Esta técnica refina los bordes, ya que tras el apartado previo los bordes pueden tener límites borrosos aún. Se debe asegurar una conclusión firme sobre si el borde se toma como tal o no. Para ello simplemente se itera alrededor de cada píxel confirmando si en las direcciones de gradiente positivo y negativo la intensidad cambia bruscamente. De esta manera se puede configurar a mano cuándo se considera que acaba el borde.

Además, se comprueba que los píxeles inmediatamente vecinos tengan el mismo comportamiento porque de no ser así se descarta el borde.

4. Añadir un segundo límite

Se realiza un segundo filtrado similar al anterior pero con parámetros más ajustados, sobre aquellos píxeles que por su bajo gradiente son posibles falsos bordes.

5. Localizar los bordes por histéresis:

Se termina suprimiendo los bordes débiles que no están conectados a bordes bien definidos.

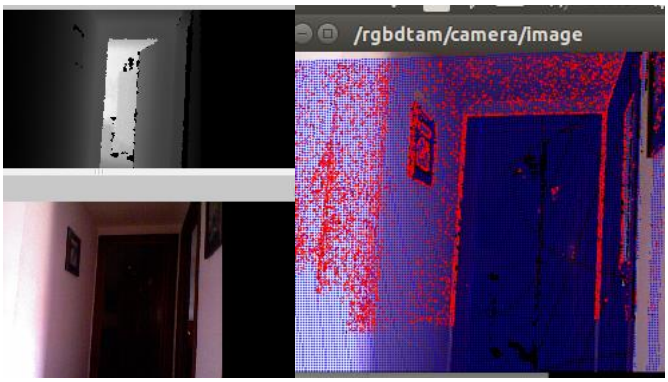


Figura 28: Resultado de la detección de bordes en RGBDTAM

Cálculo de la transformación entre frames

Para modelar la transformación T que describe la traslación y rotación del punto de vista se tienen en cuenta las dos fuentes de información como se comentaba en la introducción. La idea es minimizar una función que contenga datos sobre el error geométrico, sobre el error geométrico, y que tenga en cuenta la ganancia en brillo.

$$T = \begin{bmatrix} \exp_{SO(3)}(\delta\omega) & \delta t \\ 0_{1\times 3} & 1 \end{bmatrix}$$

$$\{\hat{T}, \hat{a}, \hat{b}\} = \arg\min_{T,a,b} r_{ph} + \lambda r_g$$

Expresión 11: Planteamiento de la transformación

Así, se tienen dos fuentes de información complementarias que minimizan el error a la vez que se está reduciendo el trabajo de procesamiento. Esto es muy útil porque el error fotométrico es inútil en entornos sin texturas y el error geométrico es inútil en entornos sin formas.

Cálculo del error fotométrico

Con la finalidad de obtener una matriz que exprese el desplazamiento (rotación y traslación) entre el frame actual y el keyframe se sigue el método directo ya explicado. En concreto la función que se aplica a los bordes de Canny es la siguiente:

$$r_{ph} = \sum_{i=1}^{n} w_p \left(\frac{\left(I_k(\pi(T_w^k p_w^i)) - aI_f(\pi(T_w^f T^{-1} p_w^i)) + b\right)^2}{\sigma_{ph}^2} \right)$$

Expresión 12: Minimización del error fotométrico

Donde se resta al primer término que es la intensidad del píxel en el keyframe actual, la intensidad del mismo punto en el frame de ese instante. Los parámetros a y b representan la estimación de los cambios en la ganancia y el brillo de la imagen. σ_{ph} es la desviación asociada al error y ω_p es una función de coste robusta llamada German-McClure cuya finalidad es eliminar la influencia de objetos dinámicos.

Para complementar esta información con una tercera coordenada se hace una estimación de la profundidad por triangulación en varios frames. Esta técnica se aplica también en el mapeo explicado en 4.3.

Cálculo del error geométrico

El segundo término de la ecuación de la transformada es el error asociado a la observación con el sensor de profundidad. Siguiendo un modelado muy similar al del apartado anterior, se minimiza la diferencia al cuadrado entre nubes de puntos dividida entre la incertidumbre del sensor. En este caso el primer término es el error que resulta de alinear la nube de puntos generada con el frame actual y minimizarlo con el mapa de profundidad observado. Igual que en la expresión 13 ω_p es la función German-McClure y el denominador es la desviación estimada del sensor.

$$r_g = \sum_{i=1}^{n} w_p \left(\frac{\left(\frac{1}{e_z{}^T T_w^f T^{-1} p_w^i} - D_f(\pi(T_w^f T^{-1} p_w^i)) \right)^2}{\sigma_g^2} \right)$$

Expresión 13: Error geométrico

A diferencia del caso anterior, ahora se trabaja con un mapa denso. Ello implica una mayor dificultad para obtener una trasformación precisa por la gran cantidad de información. Para refinar la resolución del error se implementa un algoritmo que realiza el cálculo para varios niveles de resolución de la imagen, reduciéndola desde 640x480 píxeles hasta 80x60 en cuatro pasos homogéneos. Es uno de los métodos conocidos en el mundo de la visión computacional como de tipo *pirámide,* y en este caso se incluye para intentar garantizar la operación en tiempo real.

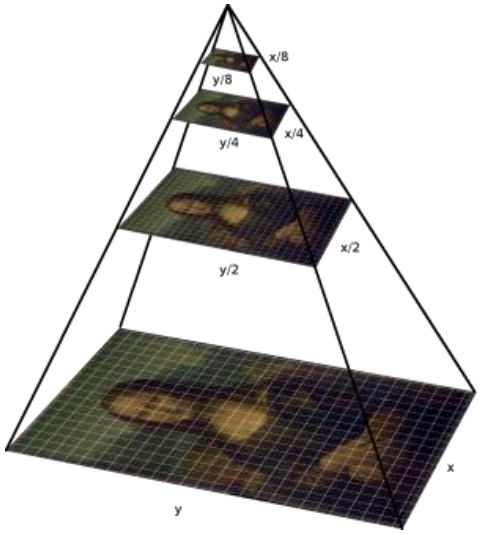


Figura 29: Reducción de resolución en pirámide

4.3. Construcción del mapa.

En RGBDTAM como en muchos otros programas la elaboración del mapa se hace combinando las dos fuentes de información que se tienen. Por un lado es necesario ir reconociendo la trayectoria para ubicar los puntos de referencia, y por otro se quiere un mapa 3D lo más detallado posible. Será necesario entonces, como se vio en el apartado 3.4, reproyectar un punto localizado en varios frames sobre un plano.

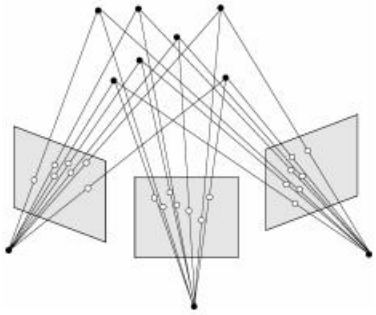


Figura 30: Esquema de la finalidad de la reproyección

La elección de crear keyframes se toma en base al porcentaje de píxeles del keyframe anterior visible en el frame actual. Cuando es demasiado bajo la precisión de la transformación cae por quedar fuera del campo visible algunos de los puntos de referencia, además de garantizar que no quedan zonas sin mapear.

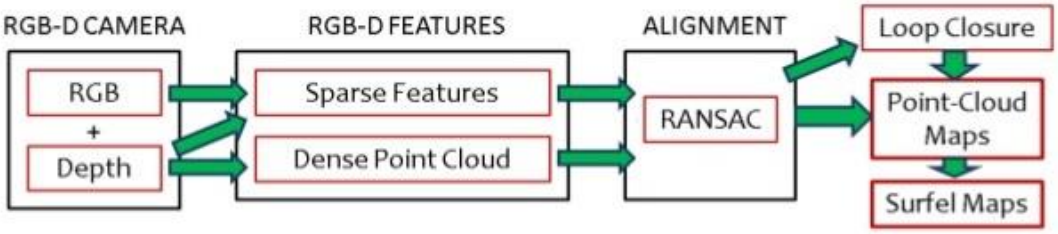


Figura 31: Algoritmo de mapeo

Para reducir el trabajo de la CPU, se ha optado por hacer un mapeo semideo. Esto quiere decir que no todos los puntos se incluyen en el mapa sino los más informativos. Lo que se consigue así es que el mapa se compute rápidamente y pueda ser utilizado en paralelo para la localización. En la figura 31 se esquematiza el algoritmo que sigue el hilo de mapeo. Con los datos de entrada se realizan dos tareas, una nube dispersa de características que se toman de referencia para asociar keyframes y una nube densa con la que se conocen en detalle las formas del entorno. Hecho esto se aplican algoritmos tipo el ya explicado RANSAC para alinear los keyframes y coserlos, lo cual será fundamental a la hora de reutilizar keyframes.

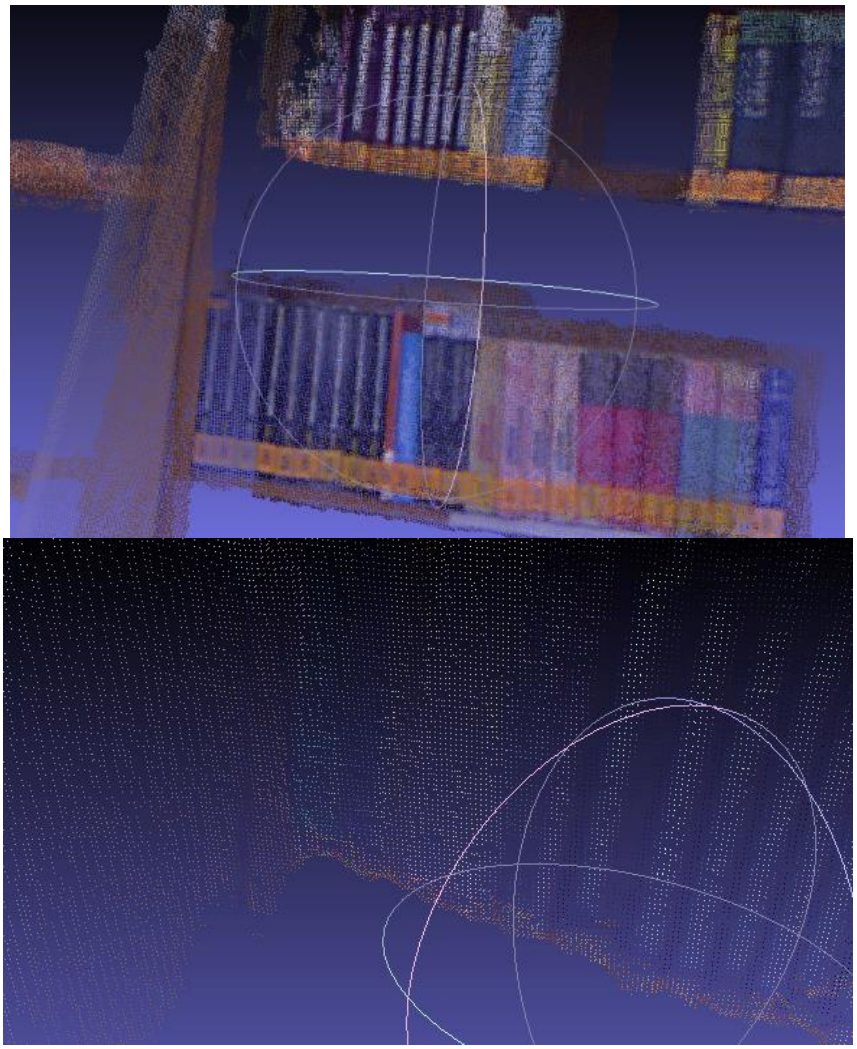


Figura 32: Detalle de la densidad del mapa

Para realizar la proyección se minimiza el error fotométrico de manera similar a como se hacía en la trayectoria:

$$r_{ph} = \sum_{o} \left\| \left(I_j \left(s_{u^*} \right) - I_o \left(G \left(s_{u^*}, T_w^j, T_w^o, \rho \right) \right) \right) \right\|_2^2 .$$

Expresión 14: Modelo del error fotométrico

Y al igual que en aquella ocasión se tiene en cuenta la transformación entre keyframe y frame actual para varios frames consecutivos. Así se obtienen varias observaciones de la misma estructura y se afianza la reproyección en el mapa.

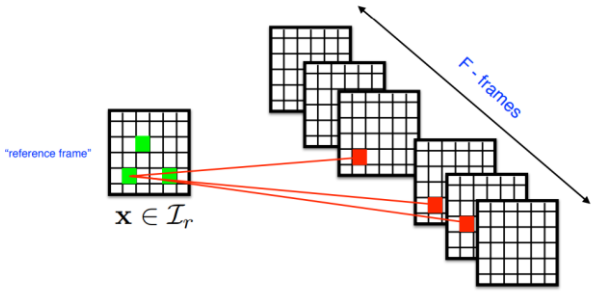


Figura 33: Objetivo de la reproyección

Una contribución importante de esta publicación es que el proceso anterior, habitual en SLAM monocular, se combina con una medida real de profundidad. Ambas mediciones tienen asociada una incertidumbre modelada previamente:

$$\rho = \frac{\sum_{j=1}^{2} \frac{\rho_j}{\sigma_j^2}}{\sum_{j=1}^{2} \frac{1}{\sigma_j^2}}, \quad \sigma = \frac{1}{\sum_{j=1}^{2} \frac{1}{\sigma_j^2}}.$$

Figura 34: Expresiones de las desviaciones de los errores

4.4. Cierre de bucles

Como se introdujo en el capítulo 3.5, el objetivo de este subsistema es reconocer cuando el robot recorre zonas ya mapeadas, y para ello se utilizan los keyframes. La finalidad principal de este subsistema es reducir al mínimo la propagación de errores al retomar referencias conocidas. Además disminuye el gasto computacional por no repetir la creación de mapas.

Al crear un keyframe se calcula su parecido con los keyframes vecinos ya que todos se han ido almacenando en memoria. Los autores establecen un parámetro umbral de 50% de correspondencias para que el keyframe sea candidato a cerrar el bucle. Estas correspondencias se hacen en base al criterio de las bolsas de palabras o vocabulario visual que ya se explicó en 3.5. La librería seleccionada es DBoW [15], creada específicamente para tratar con información obtenida por extracción de características de tipo FAST y BRIEF. Los algoritmos de DBoW necesitan un vocabulario visual entrenado para funcionar, por lo que los autores de RGBDTAM recomiendan de nuevo ORB.

En el momento que se elige un candidato, se inician las comparaciones con los keyframes más cercanos. Aplicando ORB se extraen nubes de puntos de cada uno y gracias a RANSAC se obtiene una transformación de 6 grados de libertad. Efectuando una optimización por el método de Horn [36] se llega a la matriz T_j^k que define la transformación entre los frames j y k. El siguiente paso es reproyectar un frame sobre otro y ver si se pueden establecer

asociaciones. Una vez fijado un rango en el cual un punto se considera asociado, sólo queda ver cuántos puntos se consiguen asociar y ver si son suficientes para aceptar el cierre de bucle.

4.5. ROS

ROS (Robot Operating System) es un entorno de desarrollo para robótica donde contamos con herramientas y librerías de funciones específicas para ella. No se trata de un sistema operativo al uso sino que se implementa como una colección de recursos a los que acceder desde nuestro sistema operativo habitual. Entre sus funcionalidades encontramos drivers para hardware, librerías, visualizadores, comunicaciones o manejo de paquetes. El objetivo es unificar los métodos de desarrollo y diseño de aplicaciones para robótica bajo un mismo protocolo. Así se consigue agilizar la creación de programas gracias a la filosofía colaborativa donde los desarrolladores pueden retomar las tareas donde otros las dejaron.

Existen varias colecciones de ROS, todas gratuitas, según la aplicación deseada. En RGBDTAM se ha usado Indigo por su integración más que probada con Ubuntu 14.04 y por sus recursos gráficos.

Otra de las características significativas de ROS es su mecanismo de comunicaciones, distribuido entre nodos del sistema. Un nodo es cualquier pieza de software del sistema (desde un algoritmo de SLAM, hasta un driver para el manejo de un motor). El objetivo de este método de comunicación es por una parte lograr la abstracción y reutilización de software y por otra parte adquirir independencia en cuanto a la localización del nodo. Los nodos se comunican entre ellos mediante paso de mensajes. A través de los canales llamados topics, los nodos envían y reciben mensajes basados en estructuras de datos a otros nodos suscritos al mismo topic. ROS cuenta con múltiples herramientas como rqt_graph, que representa de manera gráfica el conjunto de nodos y topics en funcionamiento, así como sus relaciones.

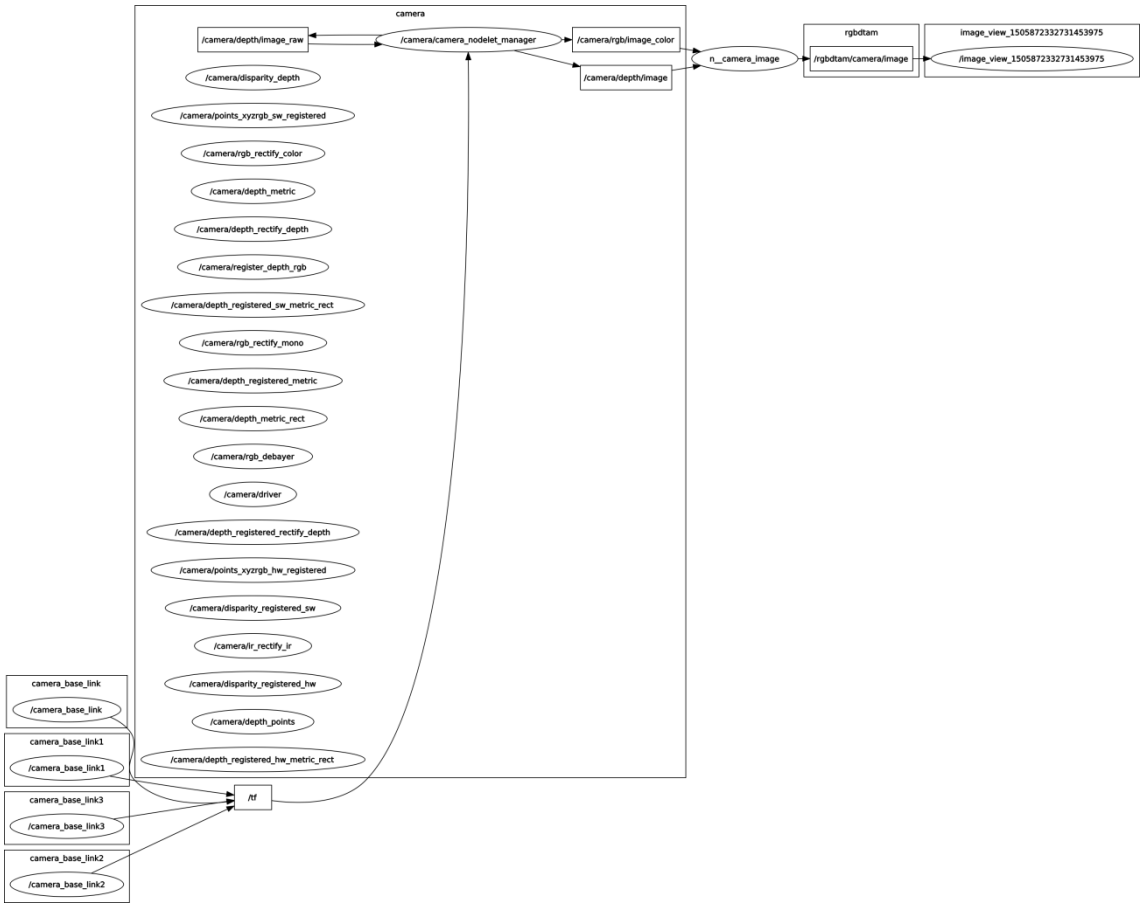


Figura 35: Gráfico rqt de nodos de RGBDTAM

Otra de las ventajas de ROS es su flexibilidad en cuanto al grado de su uso; se podría implementar por completo un sistema de control, así como la sincronización de todos sus sensores utilizando este framework y sus paquetes de software. O por el contrario, como es el caso de este proyecto, utilizarlo únicamente en una aplicación externa desde PC sin tener que instalarlo en el hardware del robot. En este caso ROS será utilizado como interfaz de comunicación entre el comunicador que recibirá las lecturas de la Kinect y los archivos C++ de RGBDTAM.

CAPÍTULO 5. Experimentos

El objetivo de este capítulo es describir las pruebas que se han realizado con el sistema RGBDTAM para conocer su potencial y analizar sus funcionalidades. Lejos de intentar cuantificar con exactitud el rendimiento que desarrolla con respecto a otros, cosa que se puede consultar en su publicación [32], el objetivo ahora es definir una serie de casos de uso y poner a prueba al sistema en ellos para demostrar en qué situaciones se comporta como debe y en qué casos muestra debilidades.

Para todo ello, se ha diseñado en primer lugar un sencillo protocolo de casos de prueba, donde se valorará si el programa está preparado para ser sometido a todo tipo de movimientos. Se intentará también buscar en qué casos el programa se pierde o desorienta y cuál es el límite de velocidad al que se puede mover el robot para que no ocurra. Esto último es el problema esencial del que se lleva hablando durante todo el documento, la rapidez en la operación. Como se ha dicho, al no contarse con los recursos para evaluar la velocidad a la que se procesa cada frame, en lugar de trabajar en obtener un dato numérico de rendimiento se extraerán una serie de conclusiones prácticas sobre la posibilidad real de aplicar este programa a aplicaciones que lo requieran.

5.1. Setup y equipo utilizado

Para todas las pruebas se ha utilizado el mismo PC, un portátil con procesador Intel i7-4510U a 3.1GHz, 8GB de RAM y 2GB de gráfica.

En el PC será necesario el sistema operativo Ubuntu para poder lanzar funciones de ROS. La versión elegida es Indigo por recomendación de los autores. Además de las librerías de ROS, serán necesarias:

- PCL, para visualización de nubes de puntos
- ORB vocabulary, para las bolsas de palabras
- BOOST, para poder lanzar hilos en paralelo

Figura 36: Imagen del sensor Kinect

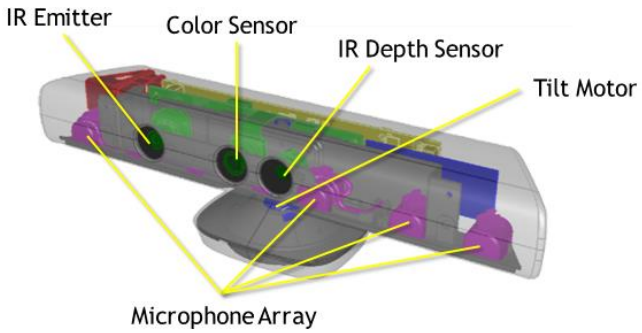


Figura 37: Arquitectura de la Kinect, imagen propiedad de Microsoft

El sensor RGB-D utilizado es la Microsoft Kinect que se puede observar en las figuras 36 y 37. Su cámara de vídeo graba entre 9 y 30 Hz a resoluciones entre 1280x1024 y 640x480 respectivamente. Cuenta con un sensor infrarrojo que graba en escala de grises a una resolución de 640x480 píxeles a 30 frames por segundo y proporciona información de profundidad con resolución de 11 bits que se traducen en unos 6 metros. Igual que con la cámara RGB la resolución de vídeo se puede aumentar bajando el ratio de frames.

A menudo se diseñan sistemas SLAM con el objetivo de escanear a mano objetos o entornos, necesitándose importantes recursos de computación de gráficos. RGBDTAM se enfoca a

otras aplicaciones y ahorra en este apartado para priorizar la localización. Por este motivo en este caso la operación en tiempo real se ve comprometida por el ruido que introduce la inestabilidad de un barrido a mano, de manera que anclamos la Kinect a un carro para realizar las exploraciones necesarias por los despachos y pasillos de la escuela.

5.2. Pruebas realizadas

A la hora de evaluar el rendimiento por métodos analíticos, los creadores de sistemas como este toman como referencia habitualmente los datasets disponibles en la web de la Universidad de Freiburg. Entre sus secuencias se pueden encontrar todo tipo de movimientos y recorridos para poner a prueba al programa. Al no interesar comparar este rendimiento en un banco de pruebas estándar, de manera similar se han creado una serie de secuencias de grabación en entornos de la universidad tratando de cubrir todos los casos de uso necesarios.

Todos estos casos de uso planteados tienen lugar en interiores bien delimitados por dos motivos. En primer lugar la Kinect debido al sensor infrarrojo con luz solar no funciona bien lo que obliga a restringir la operación a espacios cerrados, y en segundo, necesita una fuente de alimentación conectada a la red eléctrica por su nivel de consumo. Esto último tiene fácil solución utilizando una batería en lugar de la fuente en una hipotética aplicación real. Los experimentos en entornos amplios son posibles por lo tanto dentro de los rangos de la Kinect, o incluso en lugares mayores si los sensores lo permiten, pues RGBDTAM lo soportaría.

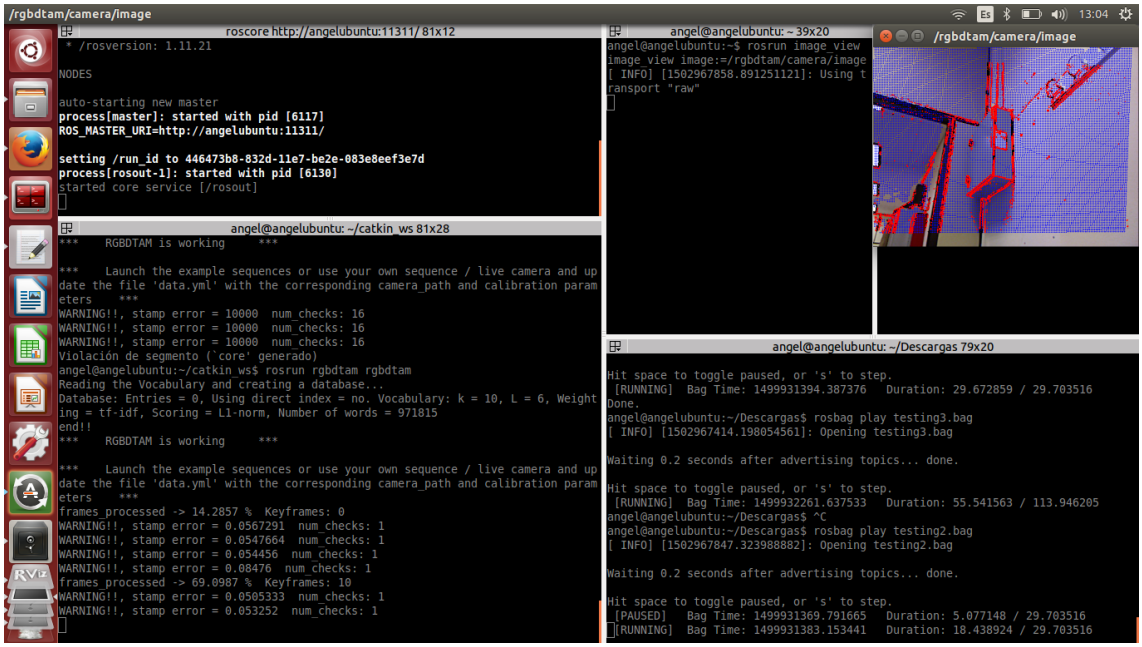


Figura 38: Entorno de trabajo en Ubuntu

Durante las pruebas la interfaz del Terminal de Ubuntu será la manera de dar las instrucciones al PC. Son necesarias entre 4 y 5 ventanas en paralelo para poder realizar todas

las tareas necesarias para lanzar RGBDTAM, que incluyen abrir ROS, conectar con la Kinect, abrir las ventanas de visualización y finalmente correr el programa.

Lo primero que se ha de hacer cada vez que se arranca una serie de pruebas es crear un puente entre ROS y Ubuntu, para lo cual se da la instrucción `roscore`, obteniéndose la pantalla de la figura 39. Roscore es una colección de nodos y programas necesarios para el funcionamiento de las librerías de ROS en Ubuntu. Lo siguiente que conviene hacer es llamar a RGBDTAM antes que a las pantallas de visualización. Esto lo haremos mediante `rosrun rgbdtam rgbdtam`, escribiendo rgbdtam dos veces puesto que la primera es el nombre de la carpeta y el segundo el del ejecutable.

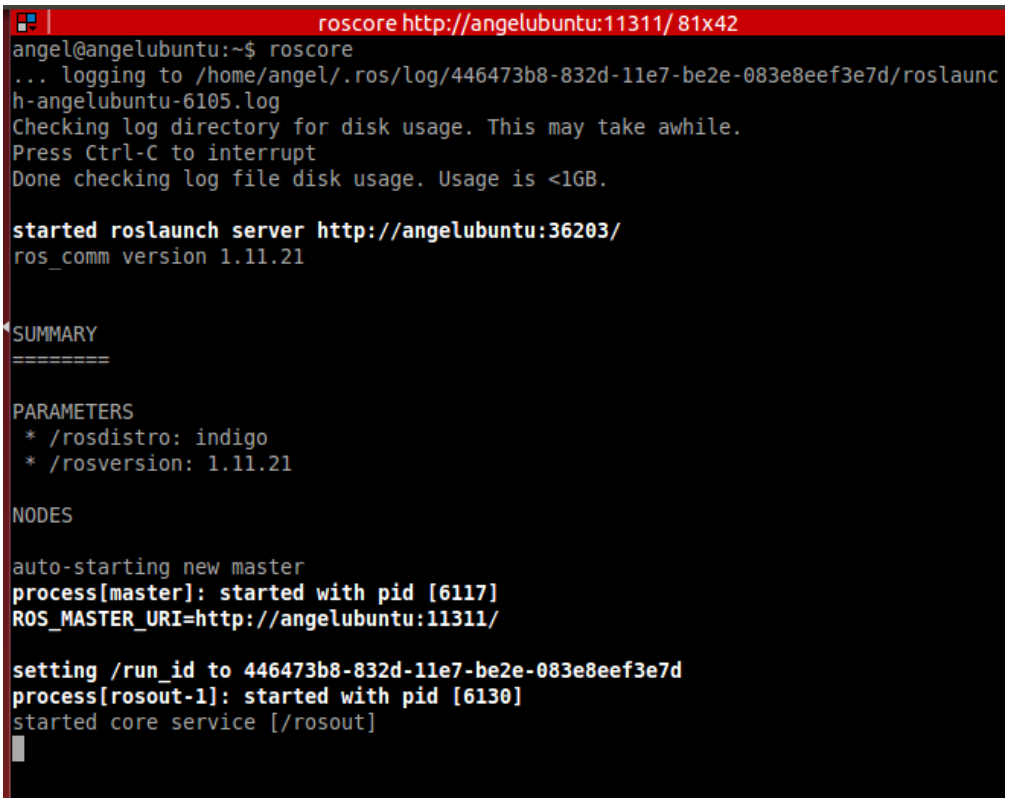


Figura 39: Pantalla del roscore

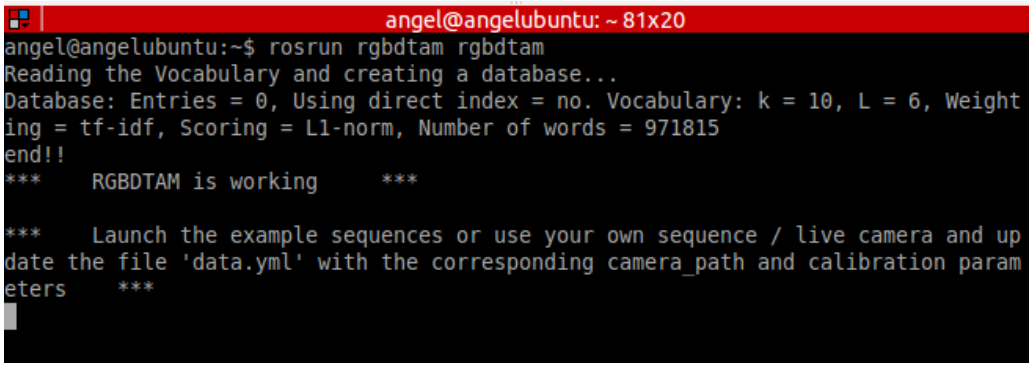


Figura 40: Confirmación de que RGBDTAM está operativo

Una vez hecho esto, llamamos a RViz para visualizar por separado los canales que graba la Kinect, es decir profundidad y RGB (figura 28), y al monitor de puntos característicos propio de RGBDTAM.

```
angel@angelubuntu: ~ 81x20
angel@angelubuntu:~$ rosrun image_view image_view image:=/rgbdtam/camera/image
```

Figura 41: Instrucción para el control de puntos de referencia

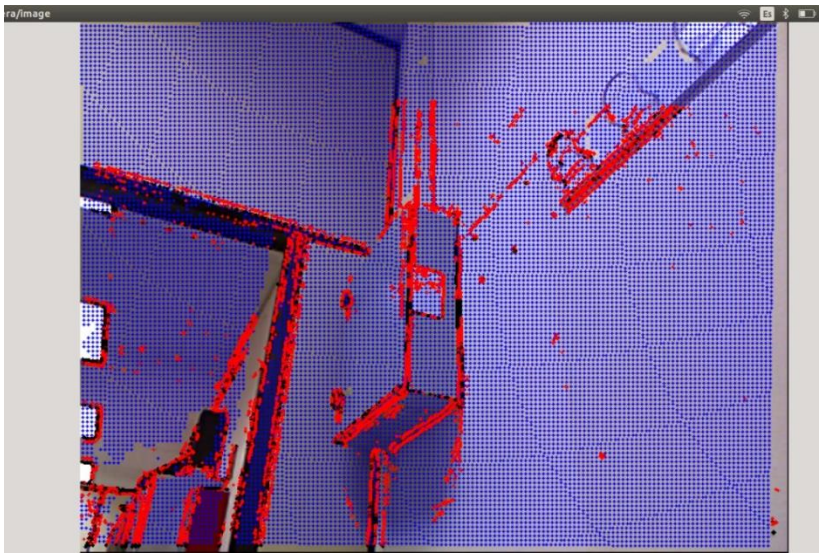


Figura 42: Visor de puntos de referencia

Con todo listo para trabajar es el momento de dar un input de vídeo al programa, que como hemos dicho vendrá de la Kinect en vivo o de una secuencia grabada previamente en un rosbag.

Un rosbag es el formato en que ROS comprime vídeos que incluyen varios topics o canales como en este caso. Para grabar un rosbag habremos mandado previamente la instrucción de la figura 43. Para ello será necesario haber conectado con la Kinect mediante su correspondiente llamada (figura 44). Si se ha optado por trabajar online basta con hacer esta misma llamada y estaremos trabajando ya.

```
angel@angelubuntu: ~ 79x20
angel@angelubuntu:~$ rosbag record camera/rgb/image_color camera/depth/image --
duration=30  -O test.bag
```

Figura 43: Creación de un rosbag

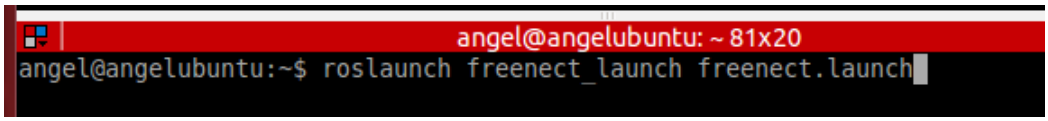


Figura 44: Llamada a conectar con la Kinect

En la ventana correspondiente a RGBDTAM iremos recibiendo además información sobre las tareas que va realizando el SW, como se ve en la figura 45.

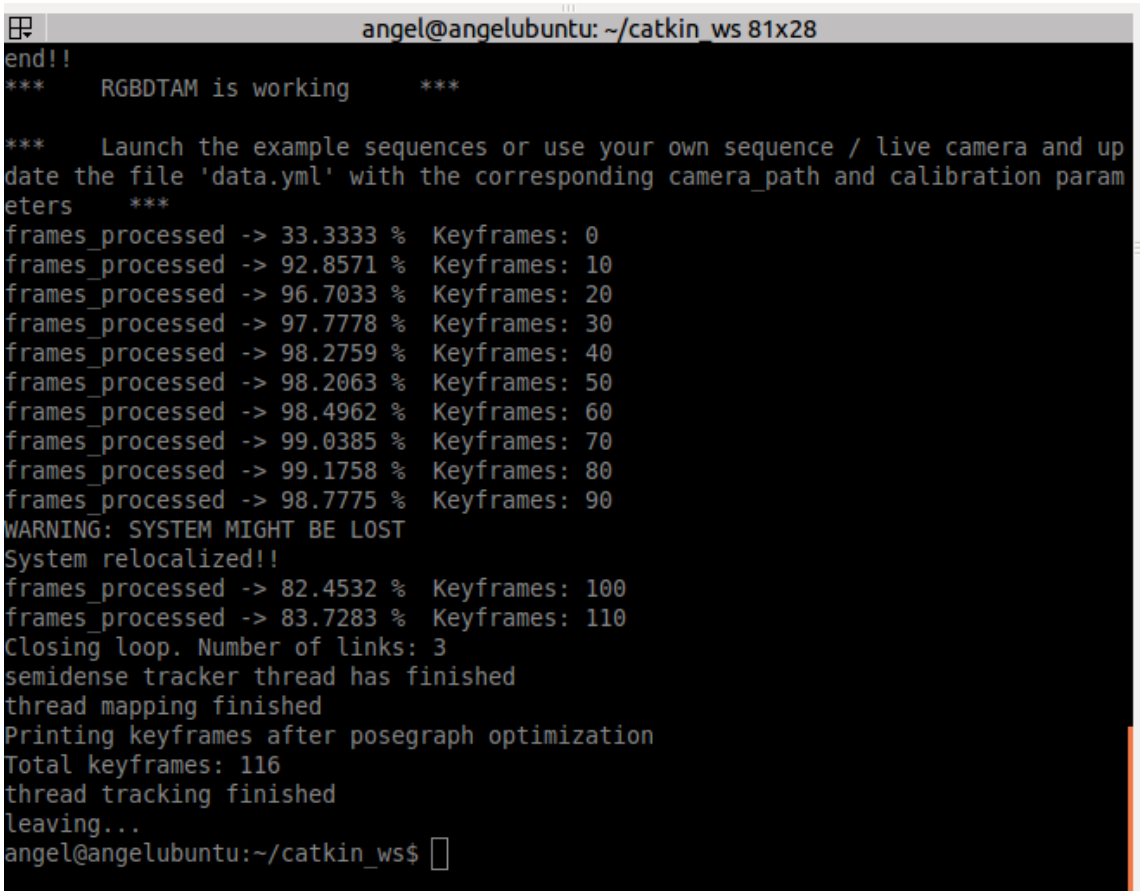


Figura 45: Monitor de tareas

Tests de operación

Las pruebas realizadas incluyen experimentos tanto online como offline y se han realizado principalmente en entornos de la escuela. Entre ellas tenemos ejemplos de laboratorios llenos de posibles puntos de referencia y pasillos homogéneos donde esto no será tan fácil, con enfoques en distintos ángulos y con trayectorias en varias direcciones. También se han hecho pruebas offline con datasets que ofrece en su web la universidad de Freiburg ya en formato rosbag.

En definitiva deberá quedar demostrado:

- Que el sistema se orienta de manera adecuada en todo tipo de situaciones,
- que puede navegar por el mapa creado

- y que puede funcionar correctamente en tiempo real.

Los casos de prueba de la localización incluyen:

- Acercamiento recto hacia la pared, traslación en z
- Giro sobre eje x (mirar hacia arriba o abajo), y (mirar a los lados) y z (suelo irregular)
- Traslación en x
- Traslación en y, en hipotética rampa o ascensor
- Giro 360º sin traslación
- Giro 360º mirando hacia abajo
- Rotación alrededor de mesa mirando a punto fijo
- Barrido en forma de S

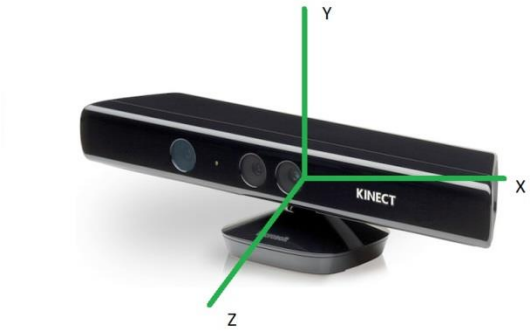


Figura 46: Sistema de referencia de la Kinect

Para probar que el mapa creado se reutiliza correctamente las pruebas que se han hecho son:

- Cerrar un bucle haciendo un recorrido que termina en el inicio, y repetirlo comprobando que reutiliza keyframes
- Cerrar un bucle haciendo un recorrido que termina en el inicio, hacer un pequeño cambio en el entorno, y repetirlo comprobando que reutiliza keyframes
- Con un bucle cerrado, giro brusco hacia zona conocida
- Con un bucle cerrado, traslación brusca sobre un eje
- Hacer un giro brusco para que el sistema se pierda. Comprobar que se inicializa en la nueva zona y que al volver a la primera zona la reconoce y cose ambas.

Todas estas pruebas se han repetido varias veces con distintos puntos de inicialización con diferentes densidades de puntos característicos.

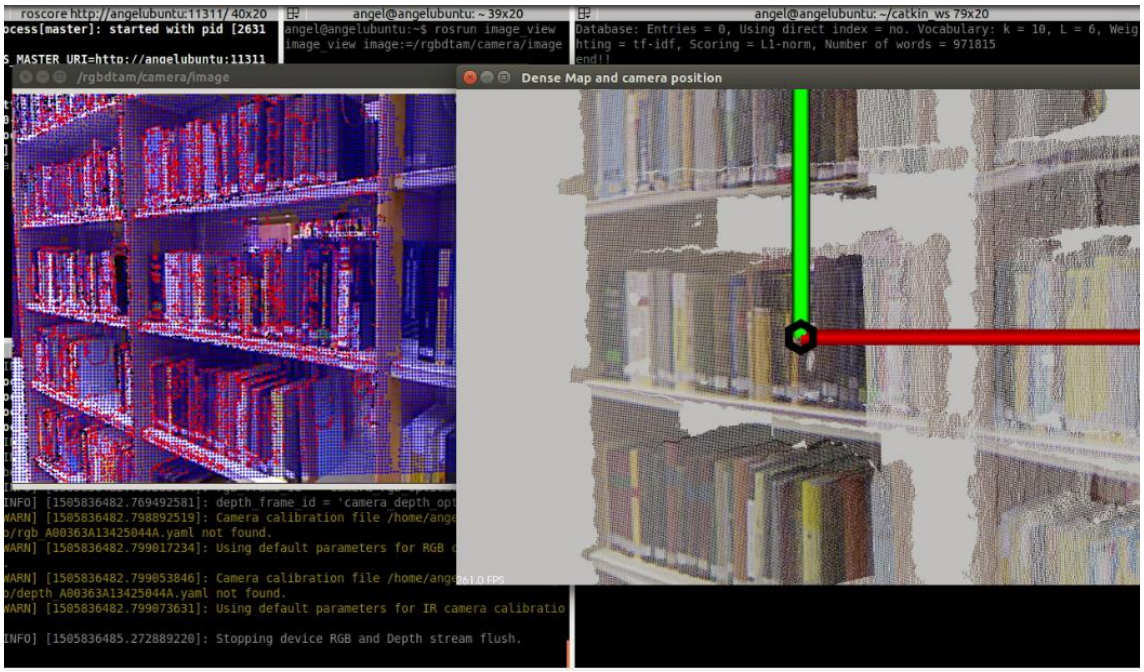


Figura 47: RGBDTAM en operación. A la izquierda la nube de puntos y a la derecha el mapa creado

Tests de rendimiento

Una de las prioridades de este proyecto es comprobar si verdaderamente la premisa que anuncian los autores de RGBDTAM se cumple para equipos de gama media, es decir, si se consigue la operación en tiempo real.

Para ello debemos lo primero identificar qué parámetros intervienen en esta ecuación. Dado que el mapa se construye con keyframes será necesario mirar en qué circunstancias podemos perder el contacto con ellos. No habrá problemas con movimientos lentos sobre una zona conocida gracias al reconocimiento de bucles, porque no se estarán consumiendo recursos en el mapeo que es la parte más pesada.

Sin embargo, recordemos que en zonas con gran cantidad de puntos de referencia aumenta la dimensión de las matrices de observación. Esto puede ralentizar los cálculos si realizamos movimientos bruscos.

Esto significa que a la pregunta de en qué punto se pierde el sistema, la respuesta será que depende del entorno puesto que cada frame requiere distinta cantidad de tiempo para procesarse. Al grabarse a 30 fps, la cifra que se busca será 33 ms de media como límite por cada frame para operar en tiempo real.

A lo largo de los experimentos anteriores se han probado distintas velocidades en el movimiento y distintos tipos de entornos con resultados diversos. Por último, se ha tenido en cuenta también qué tipo de transformación se le demanda al sistema durante las pruebas de velocidad, si traslacional, rotacional o ambas. Lo que se busca observar es en qué situaciones el sistema se desorienta, lo que significaría que no ha procesado la

transformación de puntos con respecto al keyframe a tiempo. Hecho esto se podrán extraer conclusiones sobre el tipo de operación para el que está preparado RGBDTAM y sus límites.

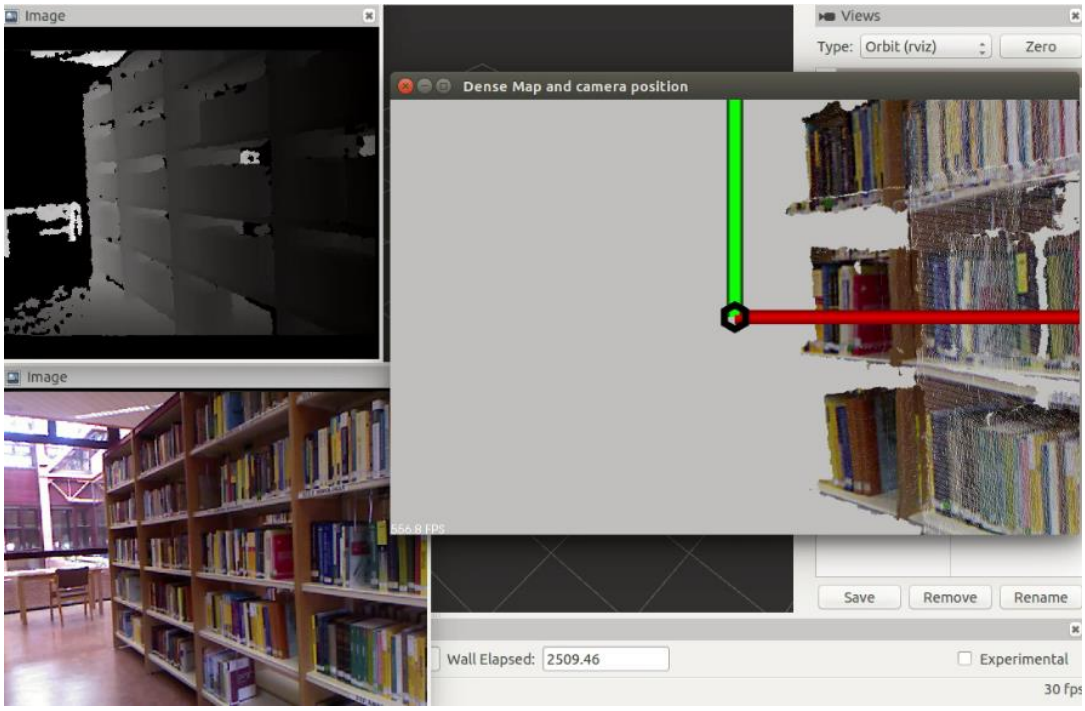


Figura 48: RGBDTAM durante la operación. A la izquierda las imágenes reales de profundidad y color, a la derecha el mapa creado

5.3. Resultados

A lo largo de las pruebas anteriormente descritas se han obtenido resultados muy variados, la mayoría favorables, aunque también se han localizado situaciones en las que el programa cierra mal bucles o reutiliza frames incorrectamente, resultando en una situación de la que sólo es posible salir reiniciando el programa.

En general se ha observado que la efectividad del hilo de mapeo y del hilo de trayectoria no tiene por qué ir pareja, lo cual es síntoma de que ha sido una gran elección optar por el formato PTAM. En concreto, habrá situaciones en las que el hilo de mapeo no pueda operar en tiempo real en la construcción del mapa, y sin embargo al seguir la localización operativa, se puede seguir navegando y almacenando información del recorrido. De hecho, se puede regresar a la zona donde se dejó de mapear y reintentarlo más despacio. Agrupando todas las pruebas anteriores por su naturaleza, los resultados han sido los siguientes:

- Las trayectorias a lo largo de los ejes de coordenadas han arrojado en general resultados positivos. La operación en tiempo real se garantiza a costa de mapeos cuya densidad baja conforme aumenta la velocidad. Por prueba y error, se ha estimado que entre 1 y 2 m/s, dependiendo de la complejidad de la estructura, se deja de poder mapear en tiempo real. Si nos movemos por zonas mapeadas, la velocidad demuestra no ser un factor decisivo salvo que sea verdaderamente brusco el cambio. En esos

casos es habitual que no se reconozca que ha habido un movimiento y tenga lugar una confusión entre la observación real y el mapa previo en la que ambos se solapan.

Figura 49: Error en la relocalización tras un impacto

- Las trayectorias a lo largo de los planos xy, xz e yz han probado funcionar correctamente bajo la misma premisa de velocidad del punto anterior. La información de distancia demuestra utilizarse correctamente para mantener coherencia entre los puntos mapeados y el punto de vista.
- Los giros se encuentran entre las situaciones más delicadas. La luz es problemática especialmente en este tipo de movimiento porque con los cambios de perspectiva cambia la forma de las sombras y se pueden alcanzar conclusiones erróneas sobre la identificación de referencias. También son las situaciones en que peores mapas obtendremos (figura 50).

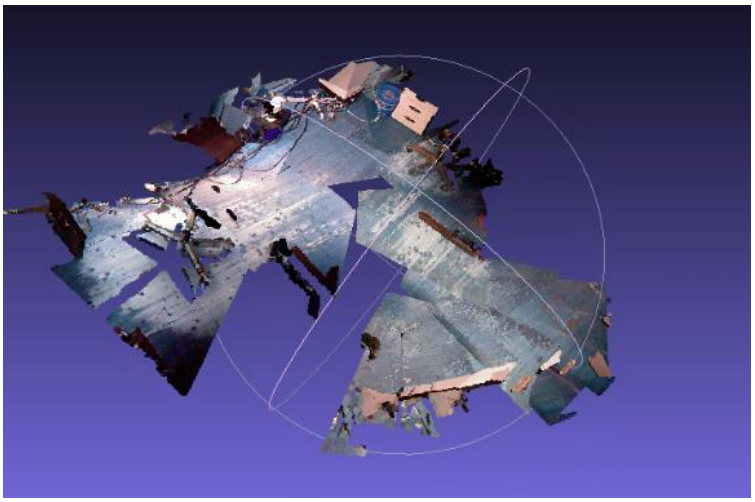


Figura 50: Reconstrucción tras una rotación sobre el eje vertical

- Las transformaciones mixtas han sido satisfactoriamente resueltas en tiempo real para velocidades bajas igual que en apartados anteriores. En secuencias grabadas a mano en las que resulta inevitable que la cámara bascule en varias direcciones mientras hacemos una traslación, se ha visto como el sistema lo resuelve correctamente. Esto es porque idealmente, este ruido que introducimos ocurre

mientras se observan zonas que ya están mapeadas y la localización en zona conocida no es un problema que comprometa la operación en tiempo real, aunque sean vibraciones o pequeñas sacudidas.

- Aparte del ejemplo que se acaba de describir, se ha probado en otro tipo de situaciones que el mapa creado se utiliza de manera correcta. Para ello basta con desandar un camino y ver que en zonas ya mapeadas la velocidad de operación es mayor. Los autores además facilitan este tipo de experimento con su interfaz porque avisan de cuando se está reutilizando un keyframe.
- Los objetos dinámicos pueden o no tener una influencia decisiva en el resultado. Si por ejemplo hemos pasado por una puerta cerrada y al volver está abierta pueden pasar dos cosas: que no se encuentren correspondencias suficientes y se cree un keyframe encima del anterior, o que sí se encuentren y se ignore el cambio en el entorno. Esto puede considerarse inofensivo en según qué aplicaciones.
- En el caso de personas caminando, la consecuencia es mucho más negativa para el proceso. Si se detecta una persona en varios puntos el sistema creyendo que es un objeto estático la incluye en el mapa de nuevo. Una vez la persona no está, como el sistema querrá utilizar el mapa para ubicarse y al acudir a esa zona no encuentra lo que espera, se pierde.
- El único caso donde realmente el programa fracasa es, lógicamente, allí donde no puede encontrar referencias. Sirvan las figuras 51 y 52 como ejemplo a lo largo de un pasillo monocromático y homogéneo.

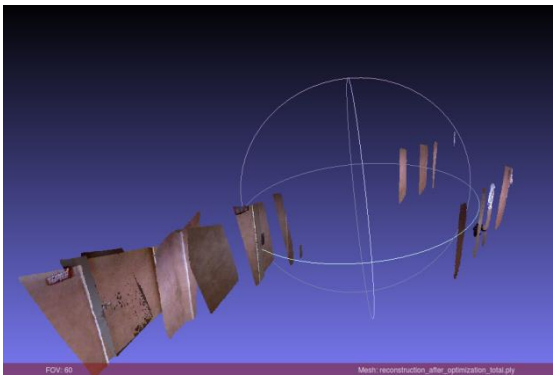


Figura 51: Intento de reconstrucción del pasillo

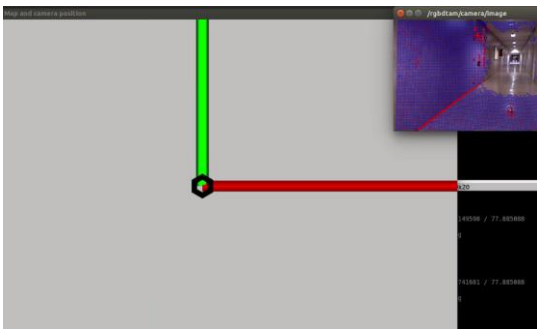


Figura 52: Ventana del mapa creado vacía junto al visor de referencias

- El cierre de bucles es de las operaciones más delicadas y donde más imprecisiones encontraremos (figura 54). El error acumulado de la reproyección y cosido de keyframes puede llevar a imprecisiones de varios centímetros. Para evitarlo lo ideal es mantener la visión paralela al suelo y a velocidades por debajo de 1 m/s. Otra consideración útil es empezar y cerrar las secuencias sin referencias tan cerca que puedan comprometer la reproyección.

```
frames_processed -> 72.5833 %  Keyframes: 50
frames_processed -> 71.9265 %  Keyframes: 60
frames_processed -> 70.9595 %  Keyframes: 70
Closing loop. Number of links: 2
REUSING previous Keyframe#:  1  Current Keyframes#:  72
Closing loop. Number of links: 6
Closing loop. Number of links: 7
Closing loop. Number of links: 11
Closing loop. Number of links: 16
semidense tracker thread has finished
thread mapping finished
Printing keyframes after posegraph optimization
Total keyframes: 77
thread tracking finished
```

Figura 53: Captura del informe de operación

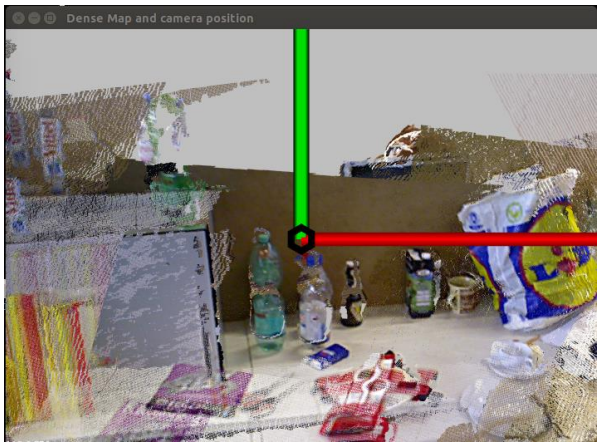


Figura 54: Bucle cerrado en falso

- La operación en tiempo real en el equipo utilizado, descrito anteriormente se cumple sólo bajo unas circunstancias muy específicas. En primer lugar la velocidad de desplazamiento ha de ser muy baja. Esto es porque la parte de localización, mientras no necesitemos crear un keyframe, puede operar sin problema. Una vez desaparecen de la vista un número suficiente de puntos característicos se crea el keyframe y mientras se opera sólo con la trayectoria durante unos milisegundos. Si la velocidad requiere la pesada tarea de crear keyframes cada poco tiempo el sistema no lo asimila y veremos que el mapa no llega a crearse.

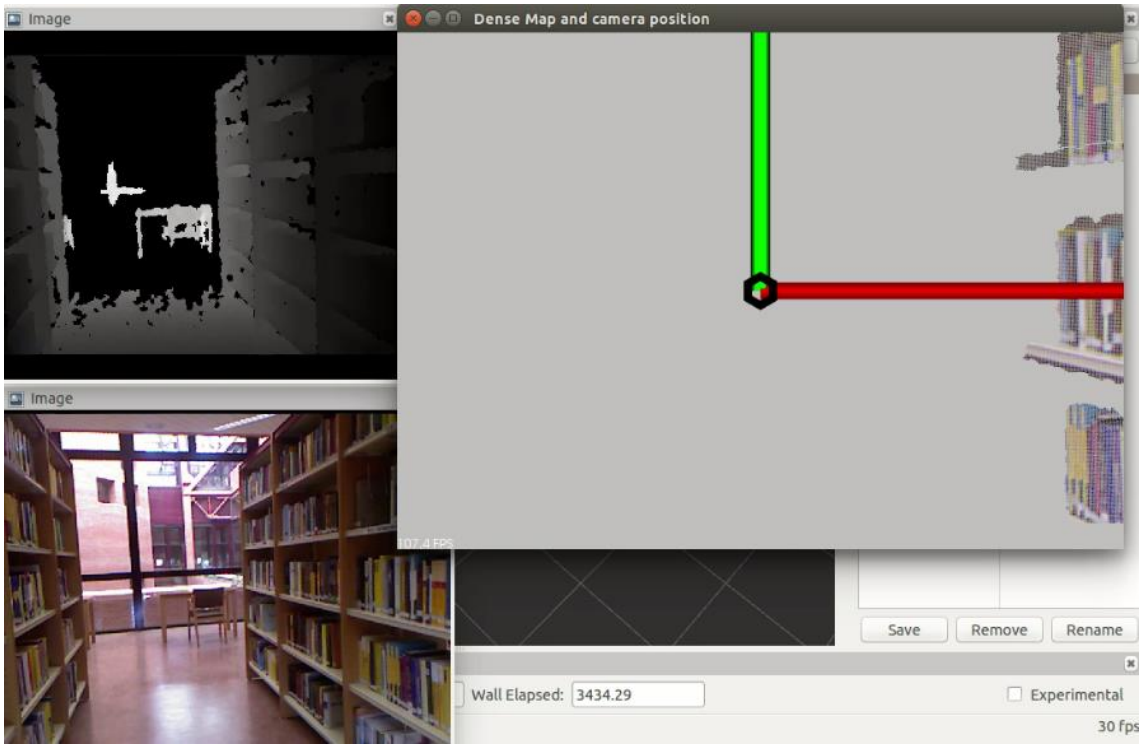


Figura 55: A la derecha, mapa no creado a tiempo.

CAPÍTULO 6. Conclusiones de los experimentos

En esta sección se pretende analizar el resultado de los test previos. Una vez estudiada la teoría sobre las bondades del método directo y vistos los experimentos, se intentará contextualizar la herramienta además de demostrar las afirmaciones que lanzan los autores de RGBDTAM en su artículo.

La primera característica que ha demostrado ser útil y cumplir su cometido es el mapeo semidenso. La cantidad de puntos que se toma para crear el mapa ha sido suficiente en todos los casos y en ningún test se ha logrado forzar un error de pérdida de orientación estando en zonas mapeadas. Queda por tanto justificado este planteamiento en pos de la eficiencia computacional.

La segunda gran contribución radica en el uso de múltiples frames consecutivos para la obtención del mapa de profundidad estimado por triangulación. Esta técnica combinada con la medición real ha arrojado resultados satisfactorios tanto en obtención de trayectorias como en mapeo. Los detalles finos se pierden ya que estas reproyecciones no dejan de ser estimaciones a las que el ruido y las rotaciones afectan de manera notoria. Considerando las posibles aplicaciones de un programa como este, no resulta dramático y las estructuras quedan plasmadas con la precisión suficiente para contextos en los que el hipotético robot no necesita interactuar de manera precisa con ella.

En cuanto a la gran incógnita de la operación en tiempo real, como ya se ha visto tiene unas restricciones importantes pero es posible. Dado que grabamos a 30 Hz, se busca un tiempo medio de procesamiento por frame de 1/30 = 33 ms. La cifra instantánea oscilará en gran medida, tanto en momentos en los que haya más cálculos de lo habitual, como durante búsquedas de bucles; como en instantes de escaso movimiento. Poniendo en conjunto todos los experimentos que realizan los autores en las secuencias de Freiburg, llegan a un resultado de 35.8 ms de media. Su equipo es ligeramente superior al utilizado en este caso (3.5 vs 3.1 GHz) y aun así vemos que alcanza la operación en tiempo real a duras penas. Son por lo tanto coherentes los resultados obtenidos en este aspecto. Esto significa que en equipos de gama más alta o allí donde el presupuesto no es un factor decisivo, este SW puede resultar realmente potente.

En definitiva, tal y como anuncia la publicación original, hemos visto que para aplicaciones que tengan lugar en interiores, y en las que la calidad del mapa creado no sea crucial, SLAM basado en el método fotométrico es una solución que puede ser verdaderamente eficiente incluso para rendimientos altos. Como ya se enunciaba en el apartado teórico, otra ventaja importante es su versatilidad en cuanto a la complejidad estructural del entorno, al combinar fuentes de información de naturaleza complementaria.

6.1. Consideraciones sobre la operación

La conclusión inmediata de todo lo anterior es que nos encontramos ante un programa que exprime al máximo los recursos del equipo en el que trabaja. Es un programa realmente pesado por lo que si queremos implementarlo con resultados satisfactorios es necesario tener muy claro la aplicación objetivo.

En primer lugar vimos que en entornos sin una complejidad estructural alta el mapeo se ve altamente afectado, pero no así la trayectoria que mantiene un nivel aceptable. Esto es importante porque si la potencia del equipo sobre el que corre lo permite, la trayectoria puede operar sin apoyarse en el mapa perfectamente.

Queda claro de las anteriores conclusiones que con PCs de inferiores características al empleado no se puede garantizar el funcionamiento en tiempo real. Aun así, recordemos que las aplicaciones de SLAM offline son variadas y con los sensores adecuados y una toma de medidas adecuada al rendimiento del equipo se puede sacar mucho partido a RGBDTAM.

6.2. Trabajo futuro

De un vistazo rápido, vemos que RGBDTAM no lleva detrás una gran labor de desarrollo sino una inteligente combinación de protocolos y algoritmos existentes. Esta viene siendo la práctica habitual en publicaciones similares, pues como se contaba en la sección sobre el estado del arte la mayoría de las innovaciones importantes sacrifican la eficiencia por el rendimiento puro.

En sus 30 años de existencia, el algoritmo SLAM y sus subsistemas no han cambiado mucho. Ha sido la tecnología de los procesadores y la mejora en los sensores los que han permitido aplicaciones más complejas y eficaces.

Esto quiere decir que tras la inmensa cantidad de investigaciones dedicadas a SLAM, es lógico creer que en breve se tocará el techo de la eficiencia. Conforme evolucione el hardware permitiendo herramientas más pesadas y robustas, será posible mejorar la polivalencia y el comportamiento en casos críticos, incluyendo algoritmos que tal vez ya existen, como el reconocimiento de personas, protocolos de reinicialización o la mejora en aplicaciones a gran escala.

CAPÍTULO 7. Normativa aplicable e impacto socio-económico

Este trabajo es un estudio principalmente teórico de una tecnología en constante mejora y crecimiento. El rango de aplicaciones crece cada día gracias a la mejora del hardware y sobre todo a la creatividad de los investigadores a la hora de resolver nuevos problemas que surgen.

Por esto mismo, no existe ni tendría demasiado sentido una normativa que regule los métodos de desarrollo de las herramientas de visión por computador. Sería responsabilidad de la compañía que vende un producto del que forma parte SLAM certificar que su producto cumple con unos determinados estándares, y esto variará enormemente con la aplicación a la que esté destinado el producto. Ya se ha visto que las aplicaciones de SLAM son muy diversas, y esto como es lógico influye en gran medida en los protocolos de validación y en los tipos de certificaciones que existen en cada industria. En general sería necesario garantizar que las debilidades que se han encontrado no suponen un problema de seguridad, y que existen métodos internos de detección de errores y de reinicialización.

Al respecto de las implicaciones sociales y económicas sí que es interesante realizar un análisis profundo puesto que la llegada de aplicaciones domésticas de SLAM al mercado abre el eterno debate de las implicaciones de la robótica y la automatización en la sociedad.

Siempre que se estudian las consecuencias de un mundo automatizado, aparecen opiniones de sectores populares que rechazan esta evolución argumentando la reducción de empleos tradicionales. Tras varias décadas de revolución tecnológica, se tiene la información suficiente para demostrar que esta inseguridad laboral no es motivo de alarma ya que siempre va a existir la supervisión del humano y que los empleos no desaparecen sino que cambian. Phil Webb, catedrático de robótica en la Universidad de Cranfield, cuenta en una entrevista que el impacto laboral es innegable a corto plazo, pero que se ha demostrado que cada robot de media acaba generando dos nuevos empleos, por un lado gracias a la mayor productividad y por otro al mayor número de trabajadores y de perfiles que hacen falta en su desarrollo. Además, existen aún áreas donde la incorporación de robots no implica sustituir al humano como la medicina. Es un mercado que se estima que ya hoy mueve mil millones de dólares al año.

Estamos ya por lo tanto inmersos en una transición durante la que la oferta de perfiles se debe ir ajustando a la demanda. Para Antonio López Peláez (Doctor en Sociología y Filosofía, Catedrático de Trabajo Social y Servicios Sociales de la UNED) el problema es que a día de hoy la nueva demanda es de perfiles más cualificados que los de los empleos que desaparecen y aquellos que pierden el empleo deben formarse en áreas radicalmente distintas, lo cual iría unido a mayor división entre clases.

Idealmente según él debería intervenir el gobierno financiando esa formación para reducir lo traumático de la situación en la mayor medida. En una publicación de investigación, el periodista Edward Luce añade, refiriéndose a la robótica a nivel industrial, que gracias a ella las empresas seguirán creciendo y ganando en productividad, pero que paradójicamente esto lleva a un salario medio inferior de la población.

Una manera habitual de atacar al problema es mirar si la sociedad está acompañando este proceso. En su libro "Race Against the Machine", Erik Brynjolfsson y Andrew McAfee señalan que hay un tremendo desfase entre un mundo que cambia tecnológicamente a una velocidad de vértigo y la educación que recibe la población. Según algunos analistas, si este desfase continúa, el desempleo masivo será inevitable, lo que plantea un problema de fondo para todo el sistema que necesita, además de productores de objetos, consumidores. También siguiendo este punto de vista existe un importante informe de la compañía de servicios financieros Merrill Lynch, que titularon “The rise of robots”. Una de sus conclusiones más contundentes sobre la robótica hace referencia también su efecto sobre el empleo: casi uno de cada dos puestos de trabajo podría desaparecer en los próximos veinte años por efecto de esta revolución, que puede incrementar la productividad en un 30% y reducir los costes laborales entre un 18% y un 33%. Entre 2015 y 2018, por ejemplo, la Federación Internacional de Robótica calcula que se venderán más de 8.000 robots “ayudantes” o “humanoides” que podrán realizar las tareas cotidianas de la oficina y del hogar. Según Merrill Lynch, el mercado de la robótica y la inteligencia artificial alcanzará un volumen de 152.000 millones de dólares (unos 142.000 millones de euros) en 2020, siendo actualmente de 32.000 millones.

Esto nos lleva a plantear las consecuencias a un nivel doméstico o de artículos tecnológicos para el gran público. Como ya vimos en el capítulo dedicado a la realidad aumentada, el hecho de que Google haya entrado a este mercado ya nos hace entender que tendrá facilidad para dominarlo, lo que significaría que aprovechando sus economías de escala, su integración con el resto de servicios y su cartera de clientes fieles, veríamos productos de realidad aumentada utilizando tecnología SLAM en la tienda de aplicaciones a corto plazo.

Si pensamos en robots de ayuda doméstica e incluso humanoides, se presenta el interrogante de si podrá el hombre y la sociedad asumir el desafío de convivir en armonía con estos seres artificiales, y sin embargo superiores. La idea de esta convivencia simultánea se remonta a Isaac Asimov (años 50), y ya en 1999 la Federación Internacional de Robótica, World Robots define un robot de servicio como “un robot que funciona total o semi autónomamente para realizar servicios útiles al bienestar de los seres humanos y al equipamiento, excluyendo las actividades de fabricación. Existe un problema de escepticismo sobre la seguridad en la robótica de servicio que lleva a diversos dilemas morales pero sobre todo el sistema legal no está totalmente preparado para asignar una responsabilidad en caso de accidente fatal.

CAPÍTULO 8. Conclusiones generales del TFG

Esta subsección tiene como único objetivo resumir la experiencia del trabajo. En la introducción se planteaban unos objetivos y unas expectativas a cumplir a través de la investigación y experimentación que conllevaba hacer este estudio.

Recapitulando: el proyecto se ha desarrollado a lo largo de tres bloques principales que son la definición de SLAM (capítulos 1 y 2), la presentación del caso de estudio (capítulos 3 y 4) y los experimentos sobre el mismo (capítulos 5 y 6).

Fijándonos en el primer bloque, es digno mencionar la cantidad de publicaciones científicas que ha sido necesario leer a fondo. Escribir sobre el estado del arte de una tecnología como SLAM resulta una tarea abrumadora ya que todos los meses se lanza algo nuevo. La mayor dificultad que entraña esta tarea es el proceso de aprendizaje necesario hasta que se entienden las mejoras que aporta cada nuevo enfoque. Tiene aquí especial importancia el conocer las aplicaciones posibles de SLAM para comprender como se beneficia cada tipo de planteamiento. Esta resulta ser la parte más interesante del proyecto al descubrirse la variedad de aplicaciones, domésticas e industriales pero por encima de todo cotidianas que empiezan a surgir para problemas resueltos con visión por computador.

Los siguientes capítulos, relacionados con la investigación más a fondo de la implementación de RGB-D SLAM, han sido el mayor reto. Los conocimientos que se aplican para desarrollar los diferentes subsistemas de SLAM, no sólo entrañan una alta complejidad, sino que en muchos casos están relacionados con áreas que el grado sólo toca de pasada. Esto ha estirado la curva de aprendizaje durante algunos meses más de lo planificado inicialmente, aunque una vez encontradas las principales claves de los algoritmos y conocidas las técnicas más empleadas, encontrar información ha sido una labor más asequible. El objetivo de esta parte era obtener una definición de la arquitectura habitual de los sistemas de este tipo, puesto que al ser los más actuales y donde más evolución ha estado habiendo en los últimos años, tienen el potencial lógico de ser los más formativos a raíz de la elaboración del proyecto.

En paralelo con el estudio anterior, se ha llevado a cabo la fase de experimentación con la herramienta RGBDTAM. La lógica de esta planificación se basa sencillamente en aprovechar los experimentos, no sólo para conocer las fortalezas y debilidades del RGB-D SLAM a nivel general, sino para buscar la relación entre los comportamientos y resultados, y los planteamientos teóricos que los autores exponen en su artículo. De esta manera se ha podido intentar justificar las innovaciones que aportan los citados autores. El hecho de que la implementación se realizara necesariamente sobre Ubuntu y ROS ha traído consigo el aprendizaje de dos herramientas de importante valor de cara al futuro profesional.

A nivel global, por lo tanto, es posible concluir que el proyecto ha cumplido con las expectativas y objetivos planteados en el primer capítulo, suponiendo un reto mayor que cualquiera visto a lo largo del grado, a la vez que resultando una experiencia enriquecedora por diversos motivos: se ha puesto a prueba la capacidad organizativa, de resolución de problemas, de búsqueda de recursos y sobre todo de adquisición de conocimientos, siendo el resultado muy satisfactorio.

CAPÍTULO 9. Referencias

[1] G. Klein, D. Murray. (2007). Parallel Tracking and Mapping for Small AR Workspaces. Active Vision Laboratory, University of Oxford.

[2] J.V. Pons. (2012). Localización y generación de mapas del entorno SLAM Universitat Politècnica de València.

[3] Riisgaard, S., & Blas, M. R. (2003). SLAM forutorial Approach to Simultaneous Localization and Mapping, 22(1-127), 126.

[4] C. Tralie (2010). SLAM and Global Navigation on the iRobot Roomba using ROS. Department of mathematics, Duke University.

[5] Grisettiyz, G., Stachniss, C., & Burgard, W. (2005, April). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on (pp. 2432-2437). IEEE.

[6] Adam Milstein (2008). Occupancy Grid Maps for Localization and Mapping, Motion Planning, Xing-Jian Jing, InTech.

[7] Engel, Jakob; Schöps, Thomas; Cremers, Daniel (2014). "European Conference on Computer Vision (ECCV)"

[8] Sarah Kudan (2016). An Introduction to Simultaneous Localisation and Mapping

[9] E. Fernández-Moral, J. González-Jiménez y V. Arévalo (2011). Partición de mapas para SLAM monocular en gran escala. Acta ROBOT 2011

[10] BBC News (2013). Los robots avanzan sobre la economía mundial.

[11] S. Koval (2014). Robots cada vez más perfectos y su impacto en la sociedad del futuro. Revista Kubernética.

[12] H. Durrant-Whyte, T. Baileyh (2006). Simultaneous Localization and Mapping Tutorial. IEEE Robotics & Automation Magazine.

[13] Moreno, L., Garrido, S., Blanco, D., & Muñoz, M. L. (2009). Differential evolution solution to the SLAM problem. Robotics and Autonomous Systems, 57(4), 441-450.

[14]. J.A. Camarena. El filtro de Kalman

[15] Gálvez-López, D., & Tardos, J. D. (2012). Bags of binary words for fast place recognition in image sequences. IEEE Transactions on Robotics, 28(5), 1188-1197.

[16] Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., & Burgard, W. (2012, May). An evaluation of the RGB-D SLAM system. In Robotics and Automation (ICRA), 2012 IEEE International Conference on (pp. 1691-1696). IEEE.

[17] Lowe, David G. "Object recognition from local scale-invariant features."Computer vision, 1999. The proceedings of the seventh IEEE international conference on. Vol. 2. Ieee, 1999.

[18] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." Computer Vision–ECCV 2006. Springer Berlin Heidelberg, 2006. 404-417.

[19] S. Ahern (2015). SLAM: Self-driving cars, AI and mapping on the move.

[20] Maurice Yap, Alessandro Bonardi, Paul Larsen and Abbie Howell (2016). Direct and Feature based Methods in SLAM. The drawbacks and benefits of the two ways of comparing images in SLAM applications.

[21] Robert Collins. Robust Estimation : RANSAC.

[22] Grisetti, G., Kummerle, R., Stachniss, C., & Burgard, W. (2010). A tutorial on graph-based SLAM. IEEE Intelligent Transportation Systems Magazine, 2(4), 31-43.

[23] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., ... & Fitzgibbon, A. (2011, October). KinectFusion: Real-time dense surface mapping and tracking. In Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on (pp. 127-136). IEEE.

[24] Tateno, K., Tombari, F., Laina, I., & Navab, N. (2017). CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. arXiv preprint arXiv:1704.03489.

[25] Ataer-Cansizoglu, E., Taguchi, Y., & Ramalingam, S. (2016, May). Pinpoint SLAM: A hybrid of 2D and 3D simultaneous localization and mapping for RGB-D sensors. In Robotics and Automation (ICRA), 2016 IEEE International Conference on (pp. 1300-1307). IEEE.

[26] S. Baker, I. Matthews (2003). Lucas-Kanade 20 Years On: A Unifying Framework. The Robotics Institute, Carnegie Mellon University

[27] Steinbrücker, F., Sturm, J., & Cremers, D. (2011, November). Real-time visual odometry from dense RGB-D images. In Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on (pp. 719-722). IEEE.

[28] Canny, J. (1986). A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, (6), 679-698.

[29] Smith, R. C., & Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. The international journal of Robotics Research, 5(4), 56-68.

[30] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Transactions on Robotics, 31(5), 1147-1163.

[31] Mur-Artal, R., & Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. IEEE Transactions on Robotics.

[32] Concha, A., & Civera, J. (2017). RGBDTAM: A Cost-Effective and Accurate RGB-D Tracking and Mapping System. arXiv preprint arXiv:1703.00754.

[33] Whelan, T., Leutenegger, S., Salas-Moreno, R., Glocker, B., & Davison, A. (2015, July). ElasticFusion: Dense SLAM without a pose graph. Robotics: Science and Systems.

[34] Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). Brief: Binary robust independent elementary features. Computer Vision–ECCV 2010, 778-792.

[35] Jakob Engel, Thomas Schops, and Daniel Cremers. LSD-SLAM: ¨ Large-Scale Direct Monocular SLAM. In European Conference on Computer Vision, pages 834–849, 2014.

[36] Berthold KP Horn (1987). Closed-form solution of absolute orientation using unit quaternions. JOSA A, 4(4):629–642.

CAPÍTULO 10. Anexos

10.1. Anexo 1: Presupuesto de elaboración TFG

Al tratarse de un estudio teórico sobre SW con una parte de experimentación, esta subsección será bastante breve.

En primer lugar el equipo con el que se ha trabajado ha sido un Acer Aspire 5 valorado en 600€ con el que ya contaba.

Se ha trabajado en entorno Ubuntu, que es un sistema operativo sin coste, al igual que el paquete de ROS.

Dentro de Ubuntu han sido necesarios diversos paquetes de librerías, drivers para Kinect y aplicaciones como Kazam para grabar la pantalla o Meshlab para abrir los mapas generados. Todo ello sin coste.

RGBDTAM se consigue en forma de archivo comprimido con todas las librerías propias y de terceros necesarias. No tiene coste pues pretende servir de base a otros investigadores.

Las librerías de terceros para implementar RGBDTAM al completo son de licencia libre y se suministran en el mismo GitHub de RGBDTAM.

El sensor Xbox Kinect ha sido prestado por el departamento de Sistemas y Automática de la Universidad.

No se ha recurrido a publicaciones de pago para realizar las investigaciones, salvo a aquellas financiadas por la Universidad y disponibles en su e-Archivo o bien entre las suscripciones de la biblioteca.

10.2. Anexo 2: Algoritmo de predicción del EKF

EKF_SLAM_Prediction$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, R_t)$**:**

2: $$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$$

3: $$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

4: $$G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$$

5: $$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + \underbrace{F_x^T \, R_t^x \, F_x}_{R_t}$$

10.3. Anexo 3: Algoritmo de corrección del EKF

EKF_SLAM_Correction

6: $Q_t = \begin{pmatrix} {\sigma_r}^2 & 0 \\ 0 & {\sigma_\phi}^2 \end{pmatrix}$

7: *for all observed features* $z_t^i = (r_t^i, \phi_t^i)^T$ *do*

8: $j = c_t^i$

9: *if landmark* j *never seen before*

10: $\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$

11: *endif*

12: $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$

13: $q = \delta^T \delta$

14: $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix}$

15: $F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 0 & \underbrace{0 \cdots 0}_{2N-2j} \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

16: $H_t^i = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x \end{pmatrix} F_{x,j}$

17: $K_t^i = \bar{\Sigma}_t \, H_t^{iT} (H_t^i \, \bar{\Sigma}_t \, H_t^{iT} + Q_t)^{-1}$

18: $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$

19: $\bar{\Sigma}_t = (I - K_t^i \, H_t^i) \, \bar{\Sigma}_t$

20: *endfor*

21: $\mu_t = \bar{\mu}_t$

22: $\Sigma_t = \bar{\Sigma}_t$

23: *return* μ_t, Σ_t

10.4. Anexo 4: Pseudocódigo para la detección de bucles cerrados

Algorithm 1 RTAB-Map

$time \leftarrow$ TIMENOW() ▷ TIMENOW() returns current time
$I_t \leftarrow$ acquired image
$L_t \leftarrow$ LOCATIONCREATION(I_t)
if z_t (of L_t) is a bad signature (using T_{bad}) **then**
  Delete L_t
else
  Insert L_t into STM, adding a neighbor link with L_{t-1}
  Weight Update of L_t in STM (using $T_{\text{similarity}}$)
  if STM's size reached its limit (T_{STM}) **then**
    Move oldest location of STM to WM
  end if
  $p(S_t|L^t) \leftarrow$Bayesian Filter Update in WM with L_t
  Loop Closure Hypothesis Selection ($S_t = i$)
  if $S_t = i$ is accepted (using T_{loop}) **then**
    Add loop closure link between L_t and L_i
  end if
  Join *trash*'s thread ▷ Thread started in TRANSFER()
  RETRIEVAL(L_i) ▷ LTM → WM
  $pTime \leftarrow$ TIMENOW() $- time$ ▷ Processing time
  if $pTime > T_{\text{time}}$ **then**
    TRANSFER() ▷ WM → LTM
  end if
end if

10.5. Anexo 5: Optimización de Gauss-Newton

Algorithm 4: Method of Gauss-Newton

input : $f : \mathbb{R}^n \to \mathbb{R}$ a function such that $f(\mathbf{x}) = \sum_{i=1}^{m} (f_i(\mathbf{x}))^2$
where all the f_i are differentiable functions from $\mathbb{R}^n$ to $\mathbb{R}$
$\mathbf{x}^{(0)}$ an initial solution
output: $\mathbf{x}^\star$, a local minimum of the cost function f.

begin
  $k \leftarrow 0$;
  while STOP-CRIT **and** $(k < k_{max})$ **do**
    $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \boldsymbol{\delta}^{(k)}$;
    with $\boldsymbol{\delta}^{(k)} = \arg\min_{\boldsymbol{\delta}} \|\mathcal{F}(\mathbf{x}^{(k)}) + \mathbf{J}_{\mathcal{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\|^2$;
    $k \leftarrow k + 1$;
  return $\mathbf{x}^{(k)}$
end